

Aerobus

v1.2

Generated by Doxygen 1.9.8

1 Introduction	1
1.1 HOW TO	1
1.1.1 Unit Test	2
1.1.2 Benchmarks	2
1.2 Structures	3
1.2.1 Predefined discrete euclidean domains	3
1.2.2 Polynomials	3
1.2.3 Known polynomials	4
1.2.4 Conway polynomials	4
1.2.5 Taylor series	4
1.3 Operations	6
1.3.1 Field of fractions	6
1.3.2 Quotient	6
1.4 Misc	7
1.4.1 Continued Fractions	7
2 Namespace Index	9
2.1 Namespace List	9
3 Concept Index	11
3.1 Concepts	11
4 Class Index	13
4.1 Class List	13
5 File Index	15
5.1 File List	15
6 Namespace Documentation	17
6.1 aerobus Namespace Reference	17
6.1.1 Detailed Description	21
6.1.2 Typedef Documentation	21
6.1.2.1 abs_t	21
6.1.2.2 addfractions_t	22
6.1.2.3 alternate_t	22
6.1.2.4 asin	22
6.1.2.5 asinh	22
6.1.2.6 atan	23
6.1.2.7 atanh	23
6.1.2.8 bell_t	23
6.1.2.9 bernoulli_t	23
6.1.2.10 combination_t	24
6.1.2.11 cos	24
6.1.2.12 cosh	24

6.1.2.13 E_fraction	24
6.1.2.14 embed_int_poly_in_fractions_t	24
6.1.2.15 exp	26
6.1.2.16 expm1	26
6.1.2.17 factorial_t	26
6.1.2.18 fpq32	26
6.1.2.19 fpq64	27
6.1.2.20 FractionField	27
6.1.2.21 gcd_t	27
6.1.2.22 geometric_sum	27
6.1.2.23 lnp1	27
6.1.2.24 make_frac_polynomial_t	28
6.1.2.25 make_int_polynomial_t	28
6.1.2.26 make_q32_t	28
6.1.2.27 make_q64_t	29
6.1.2.28 makefraction_t	29
6.1.2.29 mulfractions_t	29
6.1.2.30 pi64	29
6.1.2.31 PI_fraction	30
6.1.2.32 pow_t	30
6.1.2.33 pq64	30
6.1.2.34 q32	30
6.1.2.35 q64	30
6.1.2.36 sin	30
6.1.2.37 sinh	31
6.1.2.38 SQRT2_fraction	31
6.1.2.39 SQRT3_fraction	31
6.1.2.40 stirling_signed_t	31
6.1.2.41 stirling_unsigned_t	31
6.1.2.42 tan	32
6.1.2.43 tanh	32
6.1.2.44 taylor	32
6.1.2.45 vadd_t	33
6.1.2.46 vmul_t	33
6.1.3 Function Documentation	33
6.1.3.1 aligned_malloc()	33
6.1.3.2 field()	33
6.1.4 Variable Documentation	34
6.1.4.1 alternate_v	34
6.1.4.2 bernoulli_v	34
6.1.4.3 combination_v	34
6.1.4.4 factorial_v	35

6.2 aerobus::internal Namespace Reference	35
6.2.1 Detailed Description	38
6.2.2 Typedef Documentation	38
6.2.2.1 make_index_sequence_reverse	38
6.2.2.2 type_at_t	38
6.2.3 Function Documentation	38
6.2.3.1 index_sequence_reverse()	38
6.2.4 Variable Documentation	39
6.2.4.1 is_instantiation_of_v	39
6.3 aerobus::known_polynomials Namespace Reference	39
6.3.1 Detailed Description	39
6.3.2 Typedef Documentation	40
6.3.2.1 bernoulli	40
6.3.2.2 bernstein	40
6.3.2.3 chebyshev_T	40
6.3.2.4 chebyshev_U	41
6.3.2.5 hermite_phys	41
6.3.2.6 hermite_prob	41
6.3.2.7 laguerre	42
6.3.2.8 legendre	42
6.3.3 Enumeration Type Documentation	43
6.3.3.1 hermite_kind	43
7 Concept Documentation	45
7.1 aerobus::IsEuclideanDomain Concept Reference	45
7.1.1 Concept definition	45
7.1.2 Detailed Description	45
7.2 aerobus::IsField Concept Reference	45
7.2.1 Concept definition	45
7.2.2 Detailed Description	46
7.3 aerobus::IsRing Concept Reference	46
7.3.1 Concept definition	46
7.3.2 Detailed Description	46
8 Class Documentation	47
8.1 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E > Struct Template Reference	47
8.2 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index < 0 index > 0)> > Struct Template Reference	47
8.2.1 Member Typedef Documentation	47
8.2.1.1 type	47
8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference	48
8.3.1 Member Typedef Documentation	48

8.3.1.1 type	48
8.4 aerobus::ContinuedFraction< values > Struct Template Reference	48
8.4.1 Detailed Description	48
8.5 aerobus::ContinuedFraction< a0 > Struct Template Reference	49
8.5.1 Detailed Description	49
8.5.2 Member Typedef Documentation	49
8.5.2.1 type	49
8.5.3 Member Data Documentation	49
8.5.3.1 val	49
8.6 aerobus::ContinuedFraction< a0, rest... > Struct Template Reference	50
8.6.1 Detailed Description	50
8.6.2 Member Typedef Documentation	50
8.6.2.1 type	50
8.6.3 Member Data Documentation	51
8.6.3.1 val	51
8.7 aerobus::ConwayPolynomial Struct Reference	51
8.8 aerobus::Embed< Small, Large, E > Struct Template Reference	51
8.8.1 Detailed Description	51
8.9 aerobus::Embed< i32, i64 > Struct Reference	52
8.9.1 Detailed Description	52
8.9.2 Member Typedef Documentation	52
8.9.2.1 type	52
8.10 aerobus::Embed< polynomial< Small >, polynomial< Large > > Struct Template Reference	52
8.10.1 Detailed Description	53
8.10.2 Member Typedef Documentation	53
8.10.2.1 type	53
8.11 aerobus::Embed< q32, q64 > Struct Reference	53
8.11.1 Detailed Description	53
8.11.2 Member Typedef Documentation	54
8.11.2.1 type	54
8.12 aerobus::Embed< Quotient< Ring, X >, Ring > Struct Template Reference	54
8.12.1 Detailed Description	54
8.12.2 Member Typedef Documentation	55
8.12.2.1 type	55
8.13 aerobus::Embed< Ring, FractionField< Ring > > Struct Template Reference	55
8.13.1 Detailed Description	55
8.13.2 Member Typedef Documentation	56
8.13.2.1 type	56
8.14 aerobus::Embed< mpz< x >, i32 > Struct Template Reference	56
8.14.1 Detailed Description	56
8.14.2 Member Typedef Documentation	57
8.14.2.1 type	57

8.15 aerobus::i32 Struct Reference	57
8.15.1 Detailed Description	58
8.15.2 Member Typedef Documentation	58
8.15.2.1 add_t	58
8.15.2.2 div_t	58
8.15.2.3 eq_t	58
8.15.2.4 gcd_t	59
8.15.2.5 gt_t	59
8.15.2.6 inject_constant_t	59
8.15.2.7 inject_ring_t	59
8.15.2.8 inner_type	59
8.15.2.9 lt_t	59
8.15.2.10 mod_t	59
8.15.2.11 mul_t	59
8.15.2.12 one	60
8.15.2.13 pos_t	60
8.15.2.14 sub_t	60
8.15.2.15 zero	60
8.15.3 Member Data Documentation	60
8.15.3.1 eq_v	60
8.15.3.2 is_euclidean_domain	60
8.15.3.3 is_field	60
8.15.3.4 pos_v	61
8.16 aerobus::i64 Struct Reference	61
8.16.1 Detailed Description	62
8.16.2 Member Typedef Documentation	62
8.16.2.1 add_t	62
8.16.2.2 div_t	62
8.16.2.3 eq_t	62
8.16.2.4 gcd_t	62
8.16.2.5 gt_t	62
8.16.2.6 inject_constant_t	63
8.16.2.7 inject_ring_t	63
8.16.2.8 inner_type	63
8.16.2.9 lt_t	63
8.16.2.10 mod_t	63
8.16.2.11 mul_t	63
8.16.2.12 one	63
8.16.2.13 pos_t	64
8.16.2.14 sub_t	64
8.16.2.15 zero	64
8.16.3 Member Data Documentation	64

8.16.3.1 eq_v	64
8.16.3.2 gt_v	64
8.16.3.3 is_euclidean_domain	64
8.16.3.4 is_field	64
8.16.3.5 lt_v	65
8.16.3.6 pos_v	65
8.17 aerobus::is_prime< n > Struct Template Reference	65
8.17.1 Detailed Description	65
8.17.2 Member Data Documentation	65
8.17.2.1 value	65
8.18 aerobus::polynomial< Ring > Struct Template Reference	66
8.18.1 Detailed Description	67
8.18.2 Member Typedef Documentation	67
8.18.2.1 add_t	67
8.18.2.2 derive_t	68
8.18.2.3 div_t	68
8.18.2.4 eq_t	68
8.18.2.5 gcd_t	68
8.18.2.6 gt_t	69
8.18.2.7 inject_constant_t	69
8.18.2.8 inject_ring_t	69
8.18.2.9 lt_t	69
8.18.2.10 mod_t	70
8.18.2.11 monomial_t	70
8.18.2.12 mul_t	70
8.18.2.13 one	70
8.18.2.14 pos_t	71
8.18.2.15 simplify_t	71
8.18.2.16 sub_t	71
8.18.2.17 X	71
8.18.2.18 zero	71
8.18.3 Member Data Documentation	72
8.18.3.1 is_euclidean_domain	72
8.18.3.2 is_field	72
8.18.3.3 pos_v	72
8.19 aerobus::type_list< Ts >::pop_front Struct Reference	72
8.19.1 Detailed Description	72
8.19.2 Member Typedef Documentation	73
8.19.2.1 tail	73
8.19.2.2 type	73
8.20 aerobus::Quotient< Ring, X > Struct Template Reference	73
8.20.1 Detailed Description	74

8.20.2 Member Typedef Documentation	74
8.20.2.1 add_t	74
8.20.2.2 div_t	75
8.20.2.3 eq_t	75
8.20.2.4 inject_constant_t	75
8.20.2.5 inject_ring_t	75
8.20.2.6 mod_t	76
8.20.2.7 mul_t	76
8.20.2.8 one	76
8.20.2.9 pos_t	76
8.20.2.10 zero	77
8.20.3 Member Data Documentation	77
8.20.3.1 eq_v	77
8.20.3.2 is_euclidean_domain	77
8.20.3.3 pos_v	77
8.21 aerobus::type_list< Ts >::split< index > Struct Template Reference	78
8.21.1 Detailed Description	78
8.21.2 Member Typedef Documentation	78
8.21.2.1 head	78
8.21.2.2 tail	78
8.22 aerobus::type_list< Ts > Struct Template Reference	78
8.22.1 Detailed Description	79
8.22.2 Member Typedef Documentation	79
8.22.2.1 at	79
8.22.2.2 concat	80
8.22.2.3 insert	80
8.22.2.4 push_back	80
8.22.2.5 push_front	80
8.22.2.6 remove	81
8.22.3 Member Data Documentation	81
8.22.3.1 length	81
8.23 aerobus::type_list<> Struct Reference	81
8.23.1 Detailed Description	82
8.23.2 Member Typedef Documentation	82
8.23.2.1 concat	82
8.23.2.2 insert	82
8.23.2.3 push_back	82
8.23.2.4 push_front	82
8.23.3 Member Data Documentation	82
8.23.3.1 length	82
8.24 aerobus::i32::val< x > Struct Template Reference	82
8.24.1 Detailed Description	83

8.24.2 Member Typedef Documentation	83
8.24.2.1 enclosing_type	83
8.24.2.2 is_zero_t	83
8.24.3 Member Function Documentation	84
8.24.3.1 eval()	84
8.24.3.2 get()	84
8.24.3.3 to_string()	84
8.24.4 Member Data Documentation	84
8.24.4.1 v	84
8.25 aerobus::i64::val< x > Struct Template Reference	85
8.25.1 Detailed Description	85
8.25.2 Member Typedef Documentation	86
8.25.2.1 enclosing_type	86
8.25.2.2 inner_type	86
8.25.2.3 is_zero_t	86
8.25.3 Member Function Documentation	86
8.25.3.1 eval()	86
8.25.3.2 get()	86
8.25.3.3 to_string()	87
8.25.4 Member Data Documentation	87
8.25.4.1 v	87
8.26 aerobus::polynomial< Ring >::val< coeffN, coeffs > Struct Template Reference	87
8.26.1 Detailed Description	88
8.26.2 Member Typedef Documentation	88
8.26.2.1 aN	88
8.26.2.2 coeff_at_t	88
8.26.2.3 enclosing_type	89
8.26.2.4 is_zero_t	89
8.26.2.5 ring_type	89
8.26.2.6 strip	89
8.26.3 Member Function Documentation	89
8.26.3.1 eval()	89
8.26.3.2 to_string()	90
8.26.4 Member Data Documentation	90
8.26.4.1 degree	90
8.26.4.2 is_zero_v	90
8.27 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference	91
8.27.1 Detailed Description	91
8.27.2 Member Typedef Documentation	91
8.27.2.1 raw_t	91
8.27.2.2 type	91
8.28 aerobus::zpz< p >::val< x > Struct Template Reference	91

8.28.1 Detailed Description	92
8.28.2 Member Typedef Documentation	92
8.28.2.1 enclosing_type	92
8.28.2.2 is_zero_t	93
8.28.3 Member Function Documentation	93
8.28.3.1 eval()	93
8.28.3.2 get()	93
8.28.3.3 to_string()	93
8.28.4 Member Data Documentation	93
8.28.4.1 is_zero_v	93
8.28.4.2 v	94
8.29 aerobus::polynomial< Ring >::val< coeffN > Struct Template Reference	94
8.29.1 Detailed Description	94
8.29.2 Member Typedef Documentation	95
8.29.2.1 aN	95
8.29.2.2 coeff_at_t	95
8.29.2.3 enclosing_type	95
8.29.2.4 is_zero_t	95
8.29.2.5 ring_type	95
8.29.2.6 strip	95
8.29.3 Member Function Documentation	96
8.29.3.1 eval()	96
8.29.3.2 to_string()	96
8.29.4 Member Data Documentation	96
8.29.4.1 degree	96
8.29.4.2 is_zero_v	96
8.30 aerobus::zpz< p > Struct Template Reference	96
8.30.1 Detailed Description	98
8.30.2 Member Typedef Documentation	98
8.30.2.1 add_t	98
8.30.2.2 div_t	98
8.30.2.3 eq_t	99
8.30.2.4 gcd_t	99
8.30.2.5 gt_t	99
8.30.2.6 inject_constant_t	100
8.30.2.7 inner_type	100
8.30.2.8 lt_t	100
8.30.2.9 mod_t	100
8.30.2.10 mul_t	101
8.30.2.11 one	101
8.30.2.12 pos_t	101
8.30.2.13 sub_t	101

8.30.2.14 zero	102
8.30.3 Member Data Documentation	102
8.30.3.1 eq_v	102
8.30.3.2 gt_v	102
8.30.3.3 is_euclidean_domain	102
8.30.3.4 is_field	102
8.30.3.5 lt_v	103
8.30.3.6 pos_v	103
9 File Documentation	105
9.1 README.md File Reference	105
9.2 src/aerobus.h File Reference	105
9.3 aerobus.h	105
10 Examples	191
10.1 QuotientRing	191
10.2 type_list	191
10.3 i32::template	191
10.4 i32::add_t	192
10.5 i32::sub_t	192
10.6 i32::mul_t	192
10.7 i32::div_t	192
10.8 i32::gt_t	193
10.9 i32::eq_t	193
10.10 i32::eq_v	193
10.11 i32::gcd_t	193
10.12 i32::pos_t	194
10.13 i32::pos_v	194
10.14 i64::template	194
10.15 i64::add_t	194
10.16 i64::sub_t	195
10.17 i64::mul_t	195
10.18 i64::div_t	195
10.19 i64::mod_t	195
10.20 i64::gt_t	196
10.21 i64::lt_t	196
10.22 i64::lt_v	196
10.23 i64::eq_t	196
10.24 i64::eq_v	197
10.25 i64::gcd_t	197
10.26 i64::pos_t	197
10.27 i64::pos_v	197
10.28 polynomial	198

10.29 <code>q32::add_t</code>	198
10.30 <code>FractionField</code>	198
10.31 <code>PI_fraction::val</code>	198
10.32 <code>E_fraction::val</code>	198
Index	199

Chapter 1

Introduction

`Aerobus` is a C++-20 pure header library for general algebra on polynomials, discrete rings and associated structures.

Everything in `Aerobus` is expressed as types.

We say that again as it is the most fundamental characteristic of `Aerobus` :

Everything is expressed as types

The library serves two main purposes :

- Express algebra structures and associated operations in type arithmetic, compile-time;
- Provide portable and fast evaluation functions for polynomials.

It is designed to be 'quite easily' extensible.

Given these functions are "generated" at compile time and do not rely on inline assembly, they are actually platform independent, yielding exact same results if processors have same capabilities (such as Fused-Multiply-Add instructions).

1.1 HOW TO

- Clone or download the repository somewhere, or just download the [aerobus.h](#)
- In your code, add : `#include "aerobus.h"`
- Compile with `-std=c++20` (at least) `-I<install_location>`

`Aerobus` provides a definition for low-degree (up to 997) Conway polynomials. To use them, define `AEROBUS↔_CONWAY_IMPORTS` before including [aerobus.h](#).

1.1.1 Unit Test

Install [Cmake](#) Install a recent compiler (supporting c++20), such as MSVC, G++ or Clang++

Move to the top directory then :

```
cmake -S . -B build
cmake --build build
cd build && ctest
```

Terminal should write :

```
100% tests passed, 0 tests failed out of 48
```

Alternate way :

```
make tests
```

From top directory.

1.1.2 Benchmarks

Benchmarks are written for Intel CPUs having AVX512f and AVX512vl flags, they work only on Linux operating system using g++.

In addition of Cmake and compiler, install [OpenMP](#). Then move to top directory :

```
rm -rf build
mkdir build
cd build
cmake ..
make aerobus_benchmarks
./aerobus_benchmarks
```

results on my laptop :

```
./benchmarks_avx512.exe
[std math] 5.358e-01 Gsin/s
[std fast math] 3.389e+00 Gsin/s
[aerobus deg 1] 1.871e+01 Gsin/s
average error (vs std) : 4.36e-02
max error (vs std) : 1.50e-01
[aerobus deg 3] 1.943e+01 Gsin/s
average error (vs std) : 1.85e-04
max error (vs std) : 8.17e-04
[aerobus deg 5] 1.335e+01 Gsin/s
average error (vs std) : 6.07e-07
max error (vs std) : 3.63e-06
[aerobus deg 7] 8.634e+00 Gsin/s
average error (vs std) : 1.27e-09
max error (vs std) : 9.75e-09
[aerobus deg 9] 6.171e+00 Gsin/s
average error (vs std) : 1.89e-12
max error (vs std) : 1.78e-11
[aerobus deg 11] 4.731e+00 Gsin/s
average error (vs std) : 2.12e-15
max error (vs std) : 2.40e-14
[aerobus deg 13] 3.862e+00 Gsin/s
average error (vs std) : 3.16e-17
max error (vs std) : 3.33e-16
[aerobus deg 15] 3.359e+00 Gsin/s
average error (vs std) : 3.13e-17
max error (vs std) : 3.33e-16
[aerobus deg 17] 2.947e+00 Gsin/s
average error (vs std) : 3.13e-17
max error (vs std) : 3.33e-16
average error (vs std) : 3.13e-17
max error (vs std) : 3.33e-16
```


1.2 Structures

1.2.1 Predefined discrete euclidean domains

Aerobus predefines several simple euclidean domains, such as :

- `aerobus::i32` : integers (32 bits)
- `aerobus::i64` : integers (64 bits)
- `aerobus::zpz<p>` : integers modulo p (prime number) on 32 bits

All these types represent the Ring, meaning the algebraic structure. They have a nested type `val<i>` where `i` is a scalar native value (`int32_t` or `int64_t`) to represent actual values in the ring. They have the following "operations", required by the `IsEuclideanDomain` concept :

- `add_t` : a type (specialization of `val`), representing addition between two values
- `sub_t` : a type (specialization of `val`), representing subtraction between two values
- `mul_t` : a type (specialization of `val`), representing multiplication between two values
- `div_t` : a type (specialization of `val`), representing division between two values
- `mod_t` : a type (specialization of `val`), representing modulus between two values

and the following "elements" :

- `one` : the neutral element for multiplication, `val<1>`
- `zero` : the neutral element for addition, `val<0>`

1.2.2 Polynomials

Aerobus defines polynomials as a variadic template structure, with coefficient in an arbitrary discrete euclidean domain. As `i32` or `i64`, they are given same operations and elements, which make them a euclidean domain by themselves. Similarly, `aerobus::polynomial` represents the algebraic structure, actual values are in `aerobus::polynomial::val`.

In addition, values have an evaluation function :

```
template<typename valueRing> static constexpr valueRing eval(const valueRing& x) {...}
```

Which can be used at compile time (constexpr evaluation) or runtime.

1.2.3 Known polynomials

Aerobus predefines some well known families of polynomials, such as Hermite or Bernstein :

```
using B23 = aerobus::known_polynomials::bernstein<2, 3>; // 3X^2(1-X)
constexpr float x = B32::eval(2.0F); // -12
```

They have their coefficients either in `aerobus::i64` or `aerobus::q64`. Complete list is (but is meant to be extended):

- chebyshev_T
- chebyshev_U
- laguerre
- hermite_prob
- hermite_phys
- bernstein
- legendre
- bernoulli

1.2.4 Conway polynomials

When the tag `AEROBUS_CONWAY_IMPORTS` is defined at compile time (`-DAEROBUS_CONWAY_IMPORTS`), aerobus provides definition for all Conway polynomials $CP(p, n)$ for p up to 997 and low values for n (usually less than 10).

They can be used to construct finite fields of order p^n (\mathbb{F}_{p^n}):

```
using F2 = zpz<2>;
using PF2 = polynomial<F2>;
using F4 = Quotient<PF2, ConwayPolynomial<2, 2>::type>;
```

1.2.5 Taylor series

Aerobus provides definition for Taylor expansion of known functions. They are all templates in two parameters, degree of expansion (`size_t`) and Integers (`typename`). Coefficients then live in `Fraction<Field<Integers>>`.

They can be used and evaluated:

```
using namespace aerobus;
using aero_atanh = atanh<i64, 6>;
constexpr float val = aero_atanh::eval(0.1F); // approximation of arctanh(0.1) using taylor expansion of
degree 6
```

Exposed functions are:

- `exp`
- `expm1` $e^x - 1$
- `lnp1` $\ln(x + 1)$
- `geom` $\frac{1}{1-x}$
- `sin`

- cos
- tan
- sh
- cosh
- tanh
- asin
- acos
- acosh
- asinh
- atanh

Having the capacity of specifying the degree is very important, as users may use other formats than `float64` or `float32` which require higher or lower degree to achieve correct or acceptable precision.

It's possible to define Taylor expansion by implementing a `coeff_at` structure which must meet the following requirement :

- Being template in `Integers` (`typename`) and index (`size_t`);
- Exposing a type alias `type`, some specialization of `FractionField<Integers>::val`.

For example, to define the serie $1 + x + x^2 + x^3 + \dots$, users may write:

```
template<typename Integers, size_t i>
struct my_coeff_at {
    using type = typename FractionField<Integers>::one;
};

template<typename Integers, size_t degree>
using my_series = taylor<Integers, my_coeff_at, degree>;

static constexpr double x = my_series<i64, 3>::eval(3.0);
```

On x86-64 and CUDA platforms at least, using proper compiler directives, these functions yield very performant assembly, similar or better than standard library implementation in fast math. For example, this code:

```
double compute_expml(const size_t N, double* in, double* out) {
    using V = aerobus::expml<aerobus::i64, 13>;
    for (size_t i = 0; i < N; ++i) {
        out[i] = V::eval(in[i]);
    }
}
```

Yields this assembly (clang 17, `-mavx2 -O3`) where we can see a pile of Fused-Multiply-Add vector instructions, generated because we unrolled completely the Horner evaluation loop:

```
compute_expml(unsigned long, double const*, double*):
    lea     rax, [rdi-1]
    cmp     rax, 2
    jbe     .L5
    mov     rcx, rdi
    xor     eax, eax
    vxorpd  xmm1, xmm1, xmm1
    vbroadcastsd ymm14, QWORD PTR .LC1[rip]
    vbroadcastsd ymm13, QWORD PTR .LC3[rip]
    shr     rcx, 2
    vbroadcastsd ymm12, QWORD PTR .LC5[rip]
    vbroadcastsd ymm11, QWORD PTR .LC7[rip]
    sal     rcx, 5
    vbroadcastsd ymm10, QWORD PTR .LC9[rip]
    vbroadcastsd ymm9, QWORD PTR .LC11[rip]
    vbroadcastsd ymm8, QWORD PTR .LC13[rip]
    vbroadcastsd ymm7, QWORD PTR .LC15[rip]
    vbroadcastsd ymm6, QWORD PTR .LC17[rip]
    vbroadcastsd ymm5, QWORD PTR .LC19[rip]
    vbroadcastsd ymm4, QWORD PTR .LC21[rip]
```

```

vbroadcastsd    ymm3, QWORD PTR .LC23[rip]
vbroadcastsd    ymm2, QWORD PTR .LC25[rip]
.L3:
vmovupd ymm15, YMMWORD PTR [rsi+rax]
vmovapd ymm0, ymm15
vmadd132pd      ymm0, ymm14, ymm1
vmadd132pd      ymm0, ymm13, ymm15
vmadd132pd      ymm0, ymm12, ymm15
vmadd132pd      ymm0, ymm11, ymm15
vmadd132pd      ymm0, ymm10, ymm15
vmadd132pd      ymm0, ymm9, ymm15
vmadd132pd      ymm0, ymm8, ymm15
vmadd132pd      ymm0, ymm7, ymm15
vmadd132pd      ymm0, ymm6, ymm15
vmadd132pd      ymm0, ymm5, ymm15
vmadd132pd      ymm0, ymm4, ymm15
vmadd132pd      ymm0, ymm3, ymm15
vmadd132pd      ymm0, ymm2, ymm15
vmadd132pd      ymm0, ymm1, ymm15
vmovupd YMMWORD PTR [rdx+rax], ymm0
add    rax, 32
cmp    rcx, rax
jne    .L3
mov    rax, rdi
and    rax, -4
vzeroupper

```

1.3 Operations

1.3.1 Field of fractions

Given a set (type) satisfies the `IsEuclideanDomain` concept, `Aerobus` allows to define its `field of fractions`.

This new type is again a euclidean domain, especially a field, and therefore we can define polynomials over it.

For example, integers modulo p is not a field when p is not prime. We then can define its field of fraction and polynomials over it this way:

```

using namespace aerobus;
using ZmZ = zp<8>;
using Fzmz = FractionField<ZmZ>;
using Pfzmz = polynomial<Fzmz>;

```

The same operation would stand for any set that users would have implemented in place of `ZmZ`.

For example, we can easily define `rational functions` by taking the ring of fractions of polynomials:

```

using namespace aerobus;
using RF64 = FractionField<polynomial<q64>>;

```

Which also have an evaluation function, as polynomial do.

1.3.2 Quotient

Given a ring R , `Aerobus` provides automatic implementation for `quotient ring R/X` where X is a principal ideal generated by some element, as we know this kind of ideal is two-sided as long as R is commutative (and we assume it is).

For example, if we want R to be \mathbb{Z} represented as `aerobus::i64`, we can express arithmetic modulo 17 using:

```

using namespace aerobus;
using ZpZ = Quotient<i64, i64::val<17>>;

```

As we could have using `zp<17>`.

This is mainly used to define finite fields of order p^n using Conway polynomials but may have other applications.

1.4 Misc

1.4.1 Continued Fractions

Aerobus gives an implementation for `continued fractions`. It can be used this way:

```
using namespace aerobus;  
using T = ContinuedFraction<1,2,3,4>;  
constexpr double x = T::val;
```

As practical examples, aerobus gives continued fractions of π , e , $\sqrt{2}$ and $\sqrt{3}$:

```
constexpr double A_SQRT3 = aerobus::SQRT3_fraction::val; // 1.7320508075688772935
```


Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

aerobus	Main namespace for all publicly exposed types or functions	17
aerobus::internal	Internal implementations, subject to breaking changes without notice	35
aerobus::known_polynomials	Families of well known polynomials such as Hermite or Bernstein	39

Chapter 3

Concept Index

3.1 Concepts

Here is a list of all concepts with brief descriptions:

aerobus::IsEuclideanDomain	
Concept to express R is an euclidean domain	45
aerobus::IsField	
Concept to express R is a field	45
aerobus::IsRing	
Concept to express R is a Ring	46

Chapter 4

Class Index

4.1 Class List

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

aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E >	47
aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0 index > 0)> >	47
aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> >	48
aerobus::ContinuedFraction< values >	
Continued fraction $a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots}}$	48
aerobus::ContinuedFraction< a0 >	
Specialization for only one coefficient, technically just 'a0'	49
aerobus::ContinuedFraction< a0, rest... >	
Specialization for multiple coefficients (strictly more than one)	50
aerobus::ConwayPolynomial	51
aerobus::Embed< Small, Large, E >	
Embedding - struct forward declaration	51
aerobus::Embed< i32, i64 >	
Embeds i32 into i64	52
aerobus::Embed< polynomial< Small >, polynomial< Large > >	
Embeds polynomial<Small> into polynomial<Large>	52
aerobus::Embed< q32, q64 >	
Embeds q32 into q64	53
aerobus::Embed< Quotient< Ring, X >, Ring >	
Embeds Quotient<Ring, X> into Ring	54
aerobus::Embed< Ring, FractionField< Ring > >	
Embeds values from Ring to its field of fractions	55
aerobus::Embed< zpz< x >, i32 >	
Embeds zpz values into i32	56
aerobus::i32	
32 bits signed integers, seen as a algebraic ring with related operations	57
aerobus::i64	
64 bits signed integers, seen as a algebraic ring with related operations	61
aerobus::is_prime< n >	
Checks if n is prime	65
aerobus::polynomial< Ring >	66
aerobus::type_list< Ts >::pop_front	
Removes types from head of the list	72

aerobus::Quotient< Ring, X >	
Quotient ring by the principal ideal generated by 'X' With i32 as Ring and i32::val<2> as X, Quotient is $\mathbb{Z}/2\mathbb{Z}$	73
aerobus::type_list< Ts >::split< index >	
Splits list at index	78
aerobus::type_list< Ts >	
Empty pure template struct to handle type list	78
aerobus::type_list<>	
Specialization for empty type list	81
aerobus::i32::val< x >	
Values in i32 , again represented as types	82
aerobus::i64::val< x >	
Values in i64	85
aerobus::polynomial< Ring >::val< coeffN, coeffs >	
Values (seen as types) in polynomial ring	87
aerobus::Quotient< Ring, X >::val< V >	
Projection values in the quotient ring	91
aerobus::zpz< p >::val< x >	
Values in zpz	91
aerobus::polynomial< Ring >::val< coeffN >	
Specialization for constants	94
aerobus::zpz< p >	
Congruence classes of integers modulo p (32 bits)	96

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

src/[aerobus.h](#) 105

Chapter 6

Namespace Documentation

6.1 aerobus Namespace Reference

main namespace for all publicly exposed types or functions

Namespaces

- namespace [internal](#)
internal implementations, subject to breaking changes without notice
- namespace [known_polynomials](#)
families of well known polynomials such as Hermite or Bernstein

Classes

- struct [ContinuedFraction](#)
represents a continued fraction $a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots}}$
- struct [ContinuedFraction< a0 >](#)
Specialization for only one coefficient, technically just 'a0'.
- struct [ContinuedFraction< a0, rest... >](#)
specialization for multiple coefficients (strictly more than one)
- struct [ConwayPolynomial](#)
- struct [Embed](#)
embedding - struct forward declaration
- struct [Embed< i32, i64 >](#)
embeds i32 into i64
- struct [Embed< polynomial< Small >, polynomial< Large > >](#)
embeds polynomial<Small> into polynomial<Large>
- struct [Embed< q32, q64 >](#)
embeds q32 into q64
- struct [Embed< Quotient< Ring, X >, Ring >](#)
embeds Quotient<Ring, X> into Ring
- struct [Embed< Ring, FractionField< Ring > >](#)
embeds values from Ring to its field of fractions
- struct [Embed< zpz< x >, i32 >](#)

- embeds zpz values into [i32](#)*
- struct [i32](#)
 - 32 bits signed integers, seen as a algebraic ring with related operations*
- struct [i64](#)
 - 64 bits signed integers, seen as a algebraic ring with related operations*
- struct [is_prime](#)
 - checks if n is prime*
- struct [polynomial](#)
- struct [Quotient](#)
 - [Quotient](#) ring by the principal ideal generated by 'X' With [i32](#) as Ring and [i32::val<2>](#) as X, [Quotient](#) is $\mathbb{Z}/2\mathbb{Z}$.*
- struct [type_list](#)
 - Empty pure template struct to handle type list.*
- struct [type_list<>](#)
 - specialization for empty type list*
- struct [zpz](#)
 - congruence classes of integers modulo p (32 bits)*

Concepts

- concept [IsRing](#)
 - Concept to express R is a Ring.*
- concept [IsEuclideanDomain](#)
 - Concept to express R is an euclidean domain.*
- concept [IsField](#)
 - Concept to express R is a field.*

Typedefs

- template<typename T , typename A , typename B >
 using [gcd_t](#) = typename internal::gcd< T >::template type< A, B >
 - computes the greatest common divisor of A and B*
- template<typename... vals>
 using [vadd_t](#) = typename internal::vadd< vals... >::type
 - adds multiple values ($v_1 + v_2 + \dots + v_n$) vals must have same "enclosing_type" and "enclosing_type" must have an [add_t](#) binary operator*
- template<typename... vals>
 using [vmul_t](#) = typename internal::vmul< vals... >::type
 - multiplies multiple values ($v_1 \cdot v_2 + \dots + v_n$) vals must have same "enclosing_type" and "enclosing_type" must have an [mul_t](#) binary operator*
- template<typename val >
 using [abs_t](#) = std::conditional_t< val::enclosing_type::template pos_v< val >, val, typename val::enclosing_type::template sub_t< typename val::enclosing_type::zero, val > >
 - computes absolute value of 'val' val must be a 'value' in a Ring satisfying 'IsEuclideanDomain' concept*
- template<typename Ring >
 using [FractionField](#) = typename internal::FractionFieldImpl< Ring >::type
- using [q32](#) = [FractionField](#)< [i32](#) >
 - 32 bits rationals rationals with 32 bits numerator and denominator*
- using [fpq32](#) = [FractionField](#)< [polynomial](#)< [q32](#) > >
 - rational fractions with 32 bits rational coefficients rational fractions with rationals coefficients (32 bits numerator and denominator)*
- using [q64](#) = [FractionField](#)< [i64](#) >

- 64 bits rationals rationals with 64 bits numerator and denominator*

 - using `pi64 = polynomial< i64 >`
polynomial with 64 bits integers coefficients
 - using `pq64 = polynomial< q64 >`
polynomial with 64 bits rationals coefficients
 - using `fpq64 = FractionField< polynomial< q64 > >`
polynomial with 64 bits rational coefficients
 - template<typename Ring , typename v1 , typename v2 >
using `makefraction_t = typename FractionField< Ring >::template val< v1, v2 >`
helper type : the rational V1/V2 in the field of fractions of Ring
 - template<typename v >
using `embed_int_poly_in_fractions_t = typename Embed< polynomial< typename v::ring_type >, polynomial< FractionField< typename v::ring_type > > >::template type< v >`
embed a polynomial with integers coefficients into rational coefficients polynomials
 - template<int64_t p, int64_t q>
using `make_q64_t = typename q64::template simplify_t< typename q64::val< i64::inject_constant_t< p >, i64::inject_constant_t< q > > >`
helper type : make a fraction from numerator and denominator
 - template<int32_t p, int32_t q>
using `make_q32_t = typename q32::template simplify_t< typename q32::val< i32::inject_constant_t< p >, i32::inject_constant_t< q > > >`
helper type : make a fraction from numerator and denominator
 - template<typename Ring , typename v1 , typename v2 >
using `addfractions_t = typename FractionField< Ring >::template add_t< v1, v2 >`
helper type : adds two fractions
 - template<typename Ring , typename v1 , typename v2 >
using `mulfractions_t = typename FractionField< Ring >::template mul_t< v1, v2 >`
helper type : multiplies two fractions
 - template<typename Ring , auto... xs>
using `make_int_polynomial_t = typename polynomial< Ring >::template val< typename Ring::template inject_constant_t< xs >... >`
make a polynomial with coefficients in Ring
 - template<typename Ring , auto... xs>
using `make_frac_polynomial_t = typename polynomial< FractionField< Ring >::template val< typename FractionField< Ring >::template inject_constant_t< xs >... >`
make a polynomial with coefficients in FractionField< Ring>
 - template<typename T , size_t i>
using `factorial_t = typename internal::factorial< T, i >::type`
computes factorial(i), as type
 - template<typename T , size_t k, size_t n>
using `combination_t = typename internal::combination< T, k, n >::type`
computes binomial coefficient (k among n) as type
 - template<typename T , size_t n>
using `bernoulli_t = typename internal::bernoulli< T, n >::type`
nth bernoulli number as type in T
 - template<typename T , size_t n>
using `bell_t = typename internal::bell_helper< T, n >::type`
Bell numbers.
 - template<typename T , int k>
using `alternate_t = typename internal::alternate< T, k >::type`
(-1)^k as type in T
 - template<typename T , int n, int k>
using `stirling_signed_t = typename internal::stirling_helper< T, n, k >::type`

Stirling number of first kind (signed) – as types.

- `template<typename T, int n, int k>`
using `stirling_unsigned_t` = `abs_t< typename internal::stirling_helper< T, n, k >::type >`

Stirling number of first kind (unsigned) – as types.

- `template<typename T, typename p, size_t n>`
using `pow_t` = `typename internal::pow< T, p, n >::type`
 p^n (as 'val' type in T)
- `template<typename T, template< typename, size_t index > typename coeff_at, size_t deg>`
using `taylor` = `typename internal::make_taylor_impl< T, coeff_at, internal::make_index_sequence_reverse< deg+1 > >::type`
- `template<typename Integers, size_t deg>`
using `exp` = `taylor< Integers, internal::exp_coeff, deg >`
 e^x
- `template<typename Integers, size_t deg>`
using `expm1` = `typename polynomial< FractionField< Integers > >::template sub_t< exp< Integers, deg >, typename polynomial< FractionField< Integers > >::one >`
 $e^x - 1$
- `template<typename Integers, size_t deg>`
using `lnp1` = `taylor< Integers, internal::lnp1_coeff, deg >`
 $\ln(1 + x)$
- `template<typename Integers, size_t deg>`
using `atan` = `taylor< Integers, internal::atan_coeff, deg >`
 $\arctan(x)$
- `template<typename Integers, size_t deg>`
using `sin` = `taylor< Integers, internal::sin_coeff, deg >`
 $\sin(x)$
- `template<typename Integers, size_t deg>`
using `sinh` = `taylor< Integers, internal::sh_coeff, deg >`
 $\sinh(x)$
- `template<typename Integers, size_t deg>`
using `cosh` = `taylor< Integers, internal::cosh_coeff, deg >`
 $\cosh(x)$ *hyperbolic cosine*
- `template<typename Integers, size_t deg>`
using `cos` = `taylor< Integers, internal::cos_coeff, deg >`
 $\cos(x)$ *cosinus*
- `template<typename Integers, size_t deg>`
using `geometric_sum` = `taylor< Integers, internal::geom_coeff, deg >`
 $\frac{1}{1-x}$ *zero development of $\frac{1}{1-x}$*
- `template<typename Integers, size_t deg>`
using `asin` = `taylor< Integers, internal::asin_coeff, deg >`
 $\arcsin(x)$ *arc sinus*
- `template<typename Integers, size_t deg>`
using `asinh` = `taylor< Integers, internal::asinh_coeff, deg >`
 $\operatorname{arcsinh}(x)$ *arc hyperbolic sinus*
- `template<typename Integers, size_t deg>`
using `atanh` = `taylor< Integers, internal::atanh_coeff, deg >`
 $\operatorname{arctanh}(x)$ *arc hyperbolic tangent*
- `template<typename Integers, size_t deg>`
using `tan` = `taylor< Integers, internal::tan_coeff, deg >`
 $\tan(x)$ *tangent*
- `template<typename Integers, size_t deg>`
using `tanh` = `taylor< Integers, internal::tanh_coeff, deg >`
 $\tanh(x)$ *hyperbolic tangent*

- 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 >`
- 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 >`
- using `SQRT2_fraction = ContinuedFraction< 1, 2 >`
approximation of $\sqrt{2}$
- 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 >`
approximation of

Functions

- `template<typename T >`
`T * aligned_malloc(size_t count, size_t alignment)`
- brief Conway polynomials tparam p characteristic of the [field](#) (prime number) @tparam n degree of extension
`template< int p`

Variables

- `template<typename T , size_t i>`
`constexpr T::inner_type factorial_v = internal::factorial<T, i>::value`
computes factorial(i) as value in T
- `template<typename T , size_t k, size_t n>`
`constexpr T::inner_type combination_v = internal::combination<T, k, n>::value`
computes binomial coefficients (k among n) as value
- `template<typename FloatType , typename T , size_t n>`
`constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>`
nth bernoulli number as value in FloatType
- `template<typename T , size_t k>`
`constexpr T::inner_type alternate_v = internal::alternate<T, k>::value`
 $(-1)^k$ as value from T

6.1.1 Detailed Description

main namespace for all publicly exposed types or functions

6.1.2 Typedef Documentation

6.1.2.1 abs_t

```
template<typename val >
using aerobus::abs_t = typedef std::conditional_t< val::enclosing_type::template pos_v<val>,
val, typename val::enclosing_type::template sub_t<typename val::enclosing_type::zero, val> >
```

computes absolute value of 'val' val must be a 'value' in a Ring satisfying 'IsEuclideanDomain' concept

Template Parameters

<i>val</i>	a value in a RIng, such as <code>i64::val<-2></code>
------------	--

6.1.2.2 addfractions_t

```
template<typename Ring , typename v1 , typename v2 >
using aerobus::addfractions_t = typedef typename FractionField<Ring>::template add_t<v1, v2>
```

helper type : adds two fractions

Template Parameters

<i>Ring</i>	
<i>v1</i>	belongs to FractionField<Ring>
<i>v2</i>	belongs to FransionField<Ring>

6.1.2.3 alternate_t

```
template<typename T , int k>
using aerobus::alternate_t = typedef typename internal::alternate<T, k>::type
```

$(-1)^k$ as type in T

Template Parameters

<i>T</i>	Ring type, aerobus::i64 for example
----------	---

6.1.2.4 asin

```
template<typename Integers , size_t deg>
using aerobus::asin = typedef taylor<Integers, internal::asin_coeff, deg>
```

$\arcsin(x)$ arc sinus

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.5 asinh

```
template<typename Integers , size_t deg>
using aerobus::asinh = typedef taylor<Integers, internal::asinh_coeff, deg>
```

$\operatorname{arcsinh}(x)$ arc hyperbolic sinus

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.6 atan

```
template<typename Integers , size_t deg>
using aerobus::atan = typedef taylor<Integers, internal::atan_coeff, deg>
```

$\arctan(x)$

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.7 atanh

```
template<typename Integers , size_t deg>
using aerobus::atanh = typedef taylor<Integers, internal::atanh_coeff, deg>
```

$\operatorname{arctanh}(x)$ arc hyperbolic tangent

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.8 bell_t

```
template<typename T , size_t n>
using aerobus::bell_t = typedef typename internal::bell_helper<T, n>::type
```

Bell numbers.

Template Parameters

<i>T</i>	ring type, such as aerobus::i64
<i>n</i>	index

6.1.2.9 bernoulli_t

```
template<typename T , size_t n>
using aerobus::bernoulli_t = typedef typename internal::bernoulli<T, n>::type
```

n th bernoulli number as type in T

Template Parameters

<i>T</i>	Ring type (i64)
<i>n</i>	

6.1.2.10 combination_t

```
template<typename T , size_t k, size_t n>
using aerobus::combination_t = typedef typename internal::combination<T, k, n>::type
```

computes binomial coefficient (k among n) as type

Template Parameters

<i>T</i>	Ring type (i32 for example)
----------	--

6.1.2.11 cos

```
template<typename Integers , size_t deg>
using aerobus::cos = typedef taylor<Integers, internal::cos_coeff, deg>
```

$\cos(x)$ cosinus

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.12 cosh

```
template<typename Integers , size_t deg>
using aerobus::cosh = typedef taylor<Integers, internal::cosh_coeff, deg>
```

$\cosh(x)$ hyperbolic cosine

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.13 E_fraction

```
using aerobus::E_fraction = typedef ContinuedFraction<2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1>
```

6.1.2.14 embed_int_poly_in_fractions_t

```
template<typename v >
using aerobus::embed_int_poly_in_fractions_t = typedef typename Embed< polynomial<typename v↔
::ring_type>, polynomial<FractionField<typename v::ring_type> >>::template type<v>
```

embed a polynomial with integers coefficients into rational coefficients polynomials

Lives in polynomial<FractionField<Ring>>

Template Parameters

<i>Ring</i>	Integers
<i>a</i>	value in polynomial<Ring>

6.1.2.15 exp

```
template<typename Integers , size_t deg>
using aerobus::exp = typedef taylor<Integers, internal::exp_coeff, deg>
```

$$e^x$$

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.16 expm1

```
template<typename Integers , size_t deg>
using aerobus::expm1 = typedef typename polynomial<FractionField<Integers> >::template sub_←
t< exp<Integers, deg>, typename polynomial<FractionField<Integers> >::one>
```

$$e^x - 1$$

Template Parameters

<i>T</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.17 factorial_t

```
template<typename T , size_t i>
using aerobus::factorial_t = typedef typename internal::factorial<T, i>::type
```

computes factorial(i), as type

Template Parameters

<i>T</i>	Ring type (e.g. i32)
<i>i</i>	

6.1.2.18 fpq32

```
using aerobus::fpq32 = typedef FractionField<polynomial<q32> >
```


rational fractions with 32 bits rational coefficients rational fractions with rationals coefficients (32 bits numerator and denominator)

6.1.2.19 fpq64

```
using aerobus::fpq64 = typedef FractionField<polynomial<q64> >
```

polynomial with 64 bits rational coefficients

6.1.2.20 FractionField

```
template<typename Ring >
using aerobus::FractionField = typedef typename internal::FractionFieldImpl<Ring>::type
```

6.1.2.21 gcd_t

```
template<typename T , typename A , typename B >
using aerobus::gcd_t = typedef typename internal::gcd<T>::template type<A, B>
```

computes the greatest common divisor or A and B

Template Parameters

<i>T</i>	Ring type (must be euclidean domain)
----------	--------------------------------------

6.1.2.22 geometric_sum

```
template<typename Integers , size_t deg>
using aerobus::geometric_sum = typedef taylor<Integers, internal::geom_coeff, deg>
```

$\frac{1}{1-x}$ zero development of $\frac{1}{1-x}$

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.23 ln1

```
template<typename Integers , size_t deg>
using aerobus::ln1 = typedef taylor<Integers, internal::ln1_coeff, deg>
```

$\ln(1 + x)$

Template Parameters

<i>T</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.24 `make_frac_polynomial_t`

```
template<typename Ring , auto... xs>
using aerobus::make_frac_polynomial_t = typedef typename polynomial<FractionField<Ring> >←
::template val< typename FractionField<Ring>::template inject_constant_t<xs>...>
```

make a polynomial with coefficients in `FractionField<Ring>`

Template Parameters

<i>Ring</i>	integers
<i>...xs</i>	values

6.1.2.25 `make_int_polynomial_t`

```
template<typename Ring , auto... xs>
using aerobus::make_int_polynomial_t = typedef typename polynomial<Ring>::template val< typename
Ring::template inject_constant_t<xs>...>
```

make a polynomial with coefficients in `Ring`

Template Parameters

<i>Ring</i>	integers
<i>...xs</i>	coefficients

6.1.2.26 `make_q32_t`

```
template<int32_t p, int32_t q>
using aerobus::make_q32_t = typedef typename q32::template simplify_t< typename q32::val<i32::inject_constant
i32::inject_constant_t<q> >>
```

helper type : make a fraction from numerator and denominator

Template Parameters

<i>p</i>	numerator
<i>q</i>	denominator

6.1.2.27 make_q64_t

```
template<int64_t p, int64_t q>
using aerobus::make_q64_t = typedef typename q64::template simplify_t< typename q64::val<i64::inject_constant<i64::inject_constant_t<q>>>
```

helper type : make a fraction from numerator and denominator

Template Parameters

<i>p</i>	numerator
<i>q</i>	denominator

6.1.2.28 makefraction_t

```
template<typename Ring , typename v1 , typename v2 >
using aerobus::makefraction_t = typedef typename FractionField<Ring>::template val<v1, v2>
```

helper type : the rational V1/V2 in the field of fractions of Ring

Template Parameters

<i>Ring</i>	the base ring
<i>v1</i>	value 1 in Ring
<i>v2</i>	value 2 in Ring

6.1.2.29 mulfractions_t

```
template<typename Ring , typename v1 , typename v2 >
using aerobus::mulfractions_t = typedef typename FractionField<Ring>::template mul_t<v1, v2>
```

helper type : multiplies two fractions

Template Parameters

<i>Ring</i>	
<i>v1</i>	belongs to FractionField<Ring>
<i>v2</i>	belongs to FranctionField<Ring>

6.1.2.30 pi64

```
using aerobus::pi64 = typedef polynomial<i64>
```

polynomial with 64 bits integers coefficients

6.1.2.31 PI_fraction

```
using aerobus::PI_fraction = typedef ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1,
14, 2, 1, 1, 2, 2, 2, 2, 1>
```

6.1.2.32 pow_t

```
template<typename T , typename p , size_t n>
using aerobus::pow_t = typedef typename internal::pow<T, p, n>::type
```

p^n (as 'val' type in T)

Template Parameters

<i>T</i>	(some ring type, such as aerobus::i64)
<i>p</i>	must be an instantiation of T::val
<i>n</i>	power

6.1.2.33 pq64

```
using aerobus::pq64 = typedef polynomial<q64>
```

polynomial with 64 bits rationals coefficients

6.1.2.34 q32

```
using aerobus::q32 = typedef FractionField<i32>
```

32 bits rationals rationals with 32 bits numerator and denominator

6.1.2.35 q64

```
using aerobus::q64 = typedef FractionField<i64>
```

64 bits rationals rationals with 64 bits numerator and denominator

6.1.2.36 sin

```
template<typename Integers , size_t deg>
using aerobus::sin = typedef taylor<Integers, internal::sin_coeff, deg>
```

$\sin(x)$

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.37 sinh

```
template<typename Integers , size_t deg>
using aerobus::sinh = typedef taylor<Integers, internal::sh_coeff, deg>
```

$\sinh(x)$

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.38 SQRT2_fraction

```
using aerobus::SQRT2_fraction = typedef ContinuedFraction<1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2>
```

approximation of $\sqrt{2}$

6.1.2.39 SQRT3_fraction

```
using aerobus::SQRT3_fraction = typedef 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>
```

approximation of

6.1.2.40 stirling_signed_t

```
template<typename T , int n, int k>
using aerobus::stirling_signed_t = typedef typename internal::stirling_helper<T, n, k>::type
```

Stirling number of first kind (signed) – as types.

Template Parameters

<i>T</i>	(ring type, such as aerobus::i64)
<i>n</i>	(integer)
<i>k</i>	(integer)

6.1.2.41 stirling_unsigned_t

```
template<typename T , int n, int k>
using aerobus::stirling_unsigned_t = typedef abs_t<typename internal::stirling_helper<T, n,
k>::type>
```

Stirling number of first kind (unsigned) – as types.

Template Parameters

T	(ring type, such as aerobus::i64)
n	(integer)
k	(integer)

6.1.2.42 `tan`

```
template<typename Integers , size_t deg>
using aerobus::tan = typedef taylor<Integers, internal::tan_coeff, deg>
```

$\tan(x)$ tangent

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.43 `tanh`

```
template<typename Integers , size_t deg>
using aerobus::tanh = typedef taylor<Integers, internal::tanh_coeff, deg>
```

$\tanh(x)$ hyperbolic tangent

Template Parameters

<i>Integers</i>	Ring type (for example i64)
<i>deg</i>	taylor approximation degree

6.1.2.44 `taylor`

```
template<typename T , template< typename, size_t index > typename coeff_at, size_t deg>
using aerobus::taylor = typedef typename internal::make_taylor_impl< T, coeff_at, internal::make\_index\_sequence
+ 1> >::type
```

Template Parameters

T	Used Ring type (aerobus::i64 for example)
$coeff_{\leftrightarrow_at}$	- implementation giving the 'value' (seen as type in <code>FractionField<T></code>)
deg	

6.1.2.45 vadd_t

```
template<typename... vals>
using aerobus::vadd_t = typedef typename internal::vadd<vals...>::type
```

adds multiple values ($v_1 + v_2 + \dots + v_n$) vals must have same "enclosing_type" and "enclosing_type" must have an add_t binary operator

Template Parameters

<i>...vals</i>	
----------------	--

6.1.2.46 vmul_t

```
template<typename... vals>
using aerobus::vmul_t = typedef typename internal::vmul<vals...>::type
```

multiplies multiple values ($v_1 + v_2 + \dots + v_n$) vals must have same "enclosing_type" and "enclosing_type" must have an mul_t binary operator

Template Parameters

<i>...vals</i>	
----------------	--

6.1.3 Function Documentation

6.1.3.1 aligned_malloc()

```
template<typename T >
T * aerobus::aligned_malloc (
    size_t count,
    size_t alignment )
```

'portable' aligned allocation of count elements of type T

Template Parameters

<i>T</i>	the type of elements to store
----------	-------------------------------

Parameters

<i>count</i>	the number of elements
<i>alignment</i>	boundary

6.1.3.2 field()

```
brief Conway polynomials tparam p characteristic of the aerobus::field (
```

```
prime number )
```

6.1.4 Variable Documentation

6.1.4.1 alternate_v

```
template<typename T , size_t k>
constexpr T::inner_type aerobus::alternate_v = internal::alternate<T, k>::value [inline],
[constexpr]
```

$(-1)^k$ as value from T

Template Parameters

<i>T</i>	Ring type, aerobus::i64 for example, then result will be an <code>int64_t</code>
----------	--

6.1.4.2 bernoulli_v

```
template<typename FloatType , typename T , size_t n>
constexpr FloatType aerobus::bernoulli_v = internal::bernoulli<T, n>::template value<FloatType> [inline], [constexpr]
```

nth bernoulli number as value in FloatType

Template Parameters

<i>FloatType</i>	(double or float for example)
<i>T</i>	(aerobus::i64 for example)
<i>n</i>	

6.1.4.3 combination_v

```
template<typename T , size_t k, size_t n>
constexpr T::inner_type aerobus::combination_v = internal::combination<T, k, n>::value [inline],
[constexpr]
```

computes binomial coefficients (k among n) as value

Template Parameters

<i>T</i>	(aerobus::i32 for example)
<i>k</i>	
<i>n</i>	

6.1.4.4 factorial_v

```
template<typename T , size_t i>
constexpr T::inner_type aerobus::factorial_v = internal::factorial<T, i>::value [inline],
[constexpr]
```

computes factorial(i) as value in T

Template Parameters

<i>T</i>	(aerobus::i64 for example)
<i>i</i>	

6.2 aerobus::internal Namespace Reference

internal implementations, subject to breaking changes without notice

Classes

- struct **_FractionField**
- struct **_FractionField**< Ring, std::enable_if_t< Ring::is_euclidean_domain > >
- struct **_is_prime**
- struct **_is_prime**< 0, i >
- struct **_is_prime**< 1, i >
- struct **_is_prime**< 2, i >
- struct **_is_prime**< 3, i >
- struct **_is_prime**< 5, i >
- struct **_is_prime**< 7, i >
- struct **_is_prime**< n, i, std::enable_if_t<(n !=2 &&n !=3 &&n % 2 !=0 &&n % 3==0)> >
- struct **_is_prime**< n, i, std::enable_if_t<(n !=2 &&n % 2==0)> >
- struct **_is_prime**< n, i, std::enable_if_t<(n % i==0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&i *i > n)> >
- struct **_is_prime**< n, i, std::enable_if_t<(n %(i+2) !=0 &&n % i !=0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&(i *i<=n))> >
- struct **_is_prime**< n, i, std::enable_if_t<(n %(i+2)==0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&i *i<=n)> >
- struct **_is_prime**< n, i, std::enable_if_t<(n >=9 &&i *i > n)> >
- struct **alternate**
- struct **alternate**< T, k, std::enable_if_t< k % 2 !=0 > >
- struct **alternate**< T, k, std::enable_if_t< k % 2==0 > >
- struct **asin_coeff**
- struct **asin_coeff_helper**
- struct **asin_coeff_helper**< T, i, std::enable_if_t<(i &1)==0 > >
- struct **asin_coeff_helper**< T, i, std::enable_if_t<(i &1)==1 > >
- struct **asinh_coeff**
- struct **asinh_coeff_helper**
- struct **asinh_coeff_helper**< T, i, std::enable_if_t<(i &1)==0 > >
- struct **asinh_coeff_helper**< T, i, std::enable_if_t<(i &1)==1 > >
- struct **atan_coeff**
- struct **atan_coeff_helper**
- struct **atan_coeff_helper**< T, i, std::enable_if_t<(i &1)==0 > >

- struct **atan_coeff_helper**< T, i, std::enable_if_t<(i &1)==1 > >
- struct **atanh_coeff**
- struct **atanh_coeff_helper**
- struct **atanh_coeff_helper**< T, i, std::enable_if_t<(i &1)==0 > >
- struct **atanh_coeff_helper**< T, i, std::enable_if_t<(i &1)==1 > >
- struct **bell_helper**
- struct **bell_helper**< T, 0 >
- struct **bell_helper**< T, 1 >
- struct **bell_helper**< T, n, std::enable_if_t<(n > 1)> >
- struct **bernoulli**
- struct **bernoulli**< T, 0 >
- struct **bernoulli_coeff**
- struct **bernoulli_helper**
- struct **bernoulli_helper**< T, accum, m, m >
- struct **bernstein_helper**
- struct **bernstein_helper**< 0, 0, l >
- struct **bernstein_helper**< i, m, l, std::enable_if_t<(m > 0) &&(i > 0) &&(i < m)> >
- struct **bernstein_helper**< i, m, l, std::enable_if_t<(m > 0) &&(i==0)> >
- struct **bernstein_helper**< i, m, l, std::enable_if_t<(m > 0) &&(i==m)> >
- struct **chebyshev_helper**
- struct **chebyshev_helper**< 1, 0, l >
- struct **chebyshev_helper**< 1, 1, l >
- struct **chebyshev_helper**< 2, 0, l >
- struct **chebyshev_helper**< 2, 1, l >
- struct **combination**
- struct **combination_helper**
- struct **combination_helper**< T, 0, n >
- struct **combination_helper**< T, k, n, std::enable_if_t<(n >=0 &&k >(n/2) &&k > 0)> >
- struct **combination_helper**< T, k, n, std::enable_if_t<(n >=0 &&k <=(n/2) &&k > 0)> >
- struct **cos_coeff**
- struct **cos_coeff_helper**
- struct **cos_coeff_helper**< T, i, std::enable_if_t<(i &1)==0 > >
- struct **cos_coeff_helper**< T, i, std::enable_if_t<(i &1)==1 > >
- struct **cosh_coeff**
- struct **cosh_coeff_helper**
- struct **cosh_coeff_helper**< T, i, std::enable_if_t<(i &1)==0 > >
- struct **cosh_coeff_helper**< T, i, std::enable_if_t<(i &1)==1 > >
- struct **exp_coeff**
- struct **factorial**
- struct **factorial**< T, 0 >
- struct **factorial**< T, x, std::enable_if_t<(x > 0)> >
- struct **FractionFieldImpl**
- struct **FractionFieldImpl**< Field, std::enable_if_t< Field::is_field > >
- struct **FractionFieldImpl**< Ring, std::enable_if_t<!Ring::is_field > >
- struct **gcd**

greatest common divisor computes the greatest common divisor exposes it in gcd<A, B>::type as long as Ring type is an integral domain

- struct **gcd**< Ring, std::enable_if_t< Ring::is_euclidean_domain > >
- struct **geom_coeff**
- struct **hermite_helper**
- struct **hermite_helper**< 0, known_polynomials::hermite_kind::physicist, l >
- struct **hermite_helper**< 0, known_polynomials::hermite_kind::probabilist, l >
- struct **hermite_helper**< 1, known_polynomials::hermite_kind::physicist, l >
- struct **hermite_helper**< 1, known_polynomials::hermite_kind::probabilist, l >

- struct **hermite_helper**< deg, known_polynomials::hermite_kind::physicist, I >
- struct **hermite_helper**< deg, known_polynomials::hermite_kind::probabilist, I >
- struct **insert_h**
- struct **is_instantiation_of**
- struct **is_instantiation_of**< TT, TT< Ts... > >
- struct **laguerre_helper**
- struct **laguerre_helper**< 0, I >
- struct **laguerre_helper**< 1, I >
- struct **legendre_helper**
- struct **legendre_helper**< 0, I >
- struct **legendre_helper**< 1, I >
- struct **lnp1_coeff**
- struct **lnp1_coeff**< T, 0 >
- struct **make_taylor_impl**
- struct **make_taylor_impl**< T, coeff_at, std::integer_sequence< size_t, Is... > >
- struct **pop_front_h**
- struct **pow**
- struct **pow**< T, p, n, std::enable_if_t< n==0 > >
- struct **pow**< T, p, n, std::enable_if_t<(n % 2==1)> >
- struct **pow**< T, p, n, std::enable_if_t<(n > 0 && n % 2==0)> >
- struct **pow_scalar**
- struct **remove_h**
- struct **sh_coeff**
- struct **sh_coeff_helper**
- struct **sh_coeff_helper**< T, i, std::enable_if_t<(i &1)==0 > >
- struct **sh_coeff_helper**< T, i, std::enable_if_t<(i &1)==1 > >
- struct **sin_coeff**
- struct **sin_coeff_helper**
- struct **sin_coeff_helper**< T, i, std::enable_if_t<(i &1)==0 > >
- struct **sin_coeff_helper**< T, i, std::enable_if_t<(i &1)==1 > >
- struct **split_h**
- struct **split_h**< 0, L1, L2 >
- struct **stirling_helper**
- struct **stirling_helper**< T, 0, 0 >
- struct **stirling_helper**< T, 0, n, std::enable_if_t<(n > 0)> >
- struct **stirling_helper**< T, n, 0, std::enable_if_t<(n > 0)> >
- struct **stirling_helper**< T, n, k, std::enable_if_t<(k > 0) && (n > 0)> >
- struct **tan_coeff**
- struct **tan_coeff_helper**
- struct **tan_coeff_helper**< T, i, std::enable_if_t<(i % 2) !=0 > >
- struct **tan_coeff_helper**< T, i, std::enable_if_t<(i % 2)==0 > >
- struct **tanh_coeff**
- struct **tanh_coeff_helper**
- struct **tanh_coeff_helper**< T, i, std::enable_if_t<(i % 2) !=0 > >
- struct **tanh_coeff_helper**< T, i, std::enable_if_t<(i % 2)==0 > >
- struct **type_at**
- struct **type_at**< 0, T, Ts... >
- struct **vadd**
- struct **vadd**< v1 >
- struct **vadd**< v1, vals... >
- struct **vmul**
- struct **vmul**< v1 >
- struct **vmul**< v1, vals... >

Typedefs

- `template<size_t i, typename... Ts>`
using `type_at_t` = `typename type_at< i, Ts... >::type`
- `template<std::size_t N>`
using `make_index_sequence_reverse` = `decltype(index_sequence_reverse(std::make_index_sequence< N >{}))`

Functions

- `template<std::size_t... Is>`
constexpr auto `index_sequence_reverse` (`std::index_sequence< Is... > const &`) -> `decltype(std::index_sequence< sizeof...(Is) - 1U - Is... >{})`

Variables

- `template<template< typename... > typename TT, typename T >`
constexpr bool `is_instantiation_of_v` = `is_instantiation_of<TT, T>::value`

6.2.1 Detailed Description

internal implementations, subject to breaking changes without notice

6.2.2 Typedef Documentation

6.2.2.1 make_index_sequence_reverse

```
template<std::size_t N>
using aerobus::internal::make_index_sequence_reverse = typedef decltype(index_sequence_reverse(std::make_index_sequence<N>{}))
```

6.2.2.2 type_at_t

```
template<size_t i, typename... Ts>
using aerobus::internal::type_at_t = typedef typename type_at<i, Ts...>::type
```

6.2.3 Function Documentation

6.2.3.1 index_sequence_reverse()

```
template<std::size_t... Is>
constexpr auto aerobus::internal::index_sequence_reverse (
    std::index_sequence< Is... > const & ) -> decltype(std::index_sequence< sizeof...(Is) - 1U - Is... >{}) [constexpr]
```

6.2.4 Variable Documentation

6.2.4.1 is_instantiation_of_v

```
template<template< typename... > typename TT, typename T >
constexpr bool aerobus::internal::is_instantiation_of_v = is_instantiation_of<TT, T>::value
[inline], [constexpr]
```

6.3 aerobus::known_polynomials Namespace Reference

families of well known polynomials such as Hermite or Bernstein

Typedefs

- template<size_t deg, typename I = aerobus::i64>
using [chebyshev_T](#) = typename internal::chebyshev_helper< 1, deg, I >::type
Chebyshev polynomials of first kind.
- template<size_t deg, typename I = aerobus::i64>
using [chebyshev_U](#) = typename internal::chebyshev_helper< 2, deg, I >::type
Chebyshev polynomials of second kind.
- template<size_t deg, typename I = aerobus::i64>
using [laguerre](#) = typename internal::laguerre_helper< deg, I >::type
Laguerre polynomials.
- template<size_t deg, typename I = aerobus::i64>
using [hermite_prob](#) = typename internal::hermite_helper< deg, [hermite_kind::probabilist](#), I >::type
Hermite polynomials - probabilist form.
- template<size_t deg, typename I = aerobus::i64>
using [hermite_phys](#) = typename internal::hermite_helper< deg, [hermite_kind::physicist](#), I >::type
Hermite polynomials - physicist form.
- template<size_t i, size_t m, typename I = aerobus::i64>
using [bernstein](#) = typename internal::bernstein_helper< i, m, I >::type
Bernstein polynomials.
- template<size_t deg, typename I = aerobus::i64>
using [legendre](#) = typename internal::legendre_helper< deg, I >::type
Legendre polynomials.
- template<size_t deg, typename I = aerobus::i64>
using [bernoulli](#) = [taylor](#)< I, internal::bernoulli_coeff< deg >::template inner, deg >
Bernoulli polynomials.

Enumerations

- enum [hermite_kind](#) { [probabilist](#) , [physicist](#) }

6.3.1 Detailed Description

families of well known polynomials such as Hermite or Bernstein

6.3.2 Typedef Documentation

6.3.2.1 bernoulli

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::bernoulli = typedef taylor<I, internal::bernoulli_coeff<deg>↵
::template inner, deg>
```

Bernoulli polynomials.

Lives in polynomial<FractionField<I>>

See also

[See in Wikipedia](#)

Template Parameters

<i>deg</i>	degree of polynomial
<i>I</i>	Integers ring (defaults to aerobus::i64)

6.3.2.2 bernstein

```
template<size_t i, size_t m, typename I = aerobus::i64>
using aerobus::known_polynomials::bernstein = typedef typename internal::bernstein_helper<i,
m, I>::type
```

Bernstein polynomials.

Lives in polynomial

See also

[See in Wikipedia](#)

Template Parameters

<i>i</i>	<i>index of polynomial (between 0 and m)</i>
<i>m</i>	<i>degree of polynomial</i>
<i>I</i>	<i>Integers ring (defaults to aerobus::i64)</i>

6.3.2.3 chebyshev_T

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::chebyshev_T = typedef typename internal::chebyshev_helper<1,
deg, I>::type
```

Chebyshev polynomials of first kind.

See also

[See in Wikipedia](#)

Template Parameters

<i>deg</i>	degree of polynomial
<i>integer</i>	rings (defaults to aerobus::i64)

6.3.2.4 chebyshev_U

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::chebyshev_U = typedef typename internal::chebyshev_helper<2,
deg, I>::type
```

Chebyshev polynomials of second kind.

Lives in polynomial

See also

[See in Wikipedia](#)

Template Parameters

<i>deg</i>	<i>degree of polynomial</i>
<i>integer</i>	<i>rings (defaults to aerobus::i64)</i>

6.3.2.5 hermite_phys

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::hermite_phys = typedef typename internal::hermite_helper<deg,
hermite_kind::physicist, I>::type
```

Hermite polynomials - physicist form.

See also

[See in Wikipedia](#)

Template Parameters

<i>deg</i>	degree of polynomial
------------	----------------------

6.3.2.6 hermite_prob

```
template<size_t deg, typename I = aerobus::i64>
```

```
using aerobus::known_polynomials::hermite_prob = typedef typename internal::hermite_helper<deg,
hermite_kind::probabilist, I>::type
```

Hermite polynomials - probabilist form.

See also

[See in Wikipedia](#)

Template Parameters

<i>deg</i>	degree of polynomial
------------	----------------------

6.3.2.7 laguerre

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::laguerre = typedef typename internal::laguerre_helper<deg,
I>::type
```

Laguerre polynomials.

Lives in polynomial<FractionField<I>>

See also

[See in Wikipedia](#)

Template Parameters

<i>deg</i>	degree of polynomial
<i>I</i>	Integers ring (defaults to aerobus::i64)

6.3.2.8 legendre

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::legendre = typedef typename internal::legendre_helper<deg,
I>::type
```

Legendre polynomials.

Lives in polynomial<FractionField<I>>

See also

[See in Wikipedia](#)

Template Parameters

<i>deg</i>	degree of polynomial
<i>I</i>	Integers Ring (defaults to aerobus::i64)

6.3.3 Enumeration Type Documentation

6.3.3.1 hermite_kind

```
enum aerobus::known_polynomials::hermite_kind
```

Enumerator

probabilist	
physicist	

Chapter 7

Concept Documentation

7.1 aerobus::IsEuclideanDomain Concept Reference

Concept to express R is an euclidean domain.

```
#include <aerobus.h>
```

7.1.1 Concept definition

```
template<typename R>
concept aerobus::IsEuclideanDomain = IsRing<R> && requires {
    typename R::template div_t<typename R::one, typename R::one>;
    typename R::template mod_t<typename R::one, typename R::one>;
    typename R::template gcd_t<typename R::one, typename R::one>;
    typename R::template eq_t<typename R::one, typename R::one>;
    typename R::template pos_t<typename R::one>;

    R::template pos_v<typename R::one> == true;

    R::is_euclidean_domain == true;
}
```

7.1.2 Detailed Description

Concept to express R is an euclidean domain.

7.2 aerobus::IsField Concept Reference

Concept to express R is a field.

```
#include <aerobus.h>
```

7.2.1 Concept definition

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

7.2.2 Detailed Description

Concept to express R is a field.

7.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring.

```
#include <aerobus.h>
```

7.3.1 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>;
}
```

7.3.2 Detailed Description

Concept to express R is a Ring.

Chapter 8

Class Documentation

8.1 `aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E >` Struct Template Reference

```
#include <aerobus.h>
```

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

- [src/aerobus.h](#)

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

```
#include <aerobus.h>
```

Public Types

- `using type = typename Ring::zero`

8.2.1 Member Typedef Documentation

8.2.1.1 `type`

```
template<typename Ring >  
template<typename coeffN >  
template<size_t index>  
using aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index<  
0||index > 0)> >::type = typename Ring::zero
```

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

- [src/aerobus.h](#)

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

```
#include <aerobus.h>
```

Public Types

- [using type = aN](#)

8.3.1 Member Typedef Documentation

8.3.1.1 type

```
template<typename Ring >
template<typename coeffN >
template<size_t index>
using aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> >::type = aN
```

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

- [src/aerobus.h](#)

8.4 aerobus::ContinuedFraction< values > Struct Template Reference

represents a continued fraction $a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots}}$

```
#include <aerobus.h>
```

8.4.1 Detailed Description

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

represents a continued fraction $a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots}}$

Template Parameters

<i>...values</i>	are int64_t
------------------	----------------

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

- [src/aerobus.h](#)

8.5 aerobus::ContinuedFraction< a0 > Struct Template Reference

Specialization for only one coefficient, technically just 'a0'.

```
#include <aerobus.h>
```

Public Types

- using [type](#) = typename q64::template inject_constant_t< a0 >
represented value as [aerobus::q64](#)

Static Public Attributes

- static constexpr double [val](#) = static_cast<double>(a0)
represented value as *double*

8.5.1 Detailed Description

```
template<int64_t a0>
struct aerobus::ContinuedFraction< a0 >
```

Specialization for only one coefficient, technically just 'a0'.

Template Parameters

<i>a0</i>	an integer int64_t
-----------	-----------------------

8.5.2 Member Typedef Documentation

8.5.2.1 type

```
template<int64_t a0>
using aerobus::ContinuedFraction< a0 >::type = typename q64::template inject_constant_t<a0>
represented value as aerobus::q64
```

8.5.3 Member Data Documentation

8.5.3.1 val

```
template<int64_t a0>
constexpr double aerobus::ContinuedFraction< a0 >::val = static_cast<double>(a0) [static],
[constexpr]
```

represented value as double

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

- [src/aerobus.h](#)

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

specialization for multiple coefficients (strictly more than one)

```
#include <aerobus.h>
```

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 > >
represented value as [aerobus::q64](#)

Static Public Attributes

- static constexpr double [val](#) = type::template get<double>()
represented value as double

8.6.1 Detailed Description

```
template<int64_t a0, int64_t... rest>
struct aerobus::ContinuedFraction< a0, rest... >
```

specialization for multiple coefficients (strictly more than one)

Template Parameters

<i>a0</i>	integer (int64_t)
<i>...rest</i>	integers (int64_t)

8.6.2 Member Typedef Documentation

8.6.2.1 type

```
template<int64_t a0, int64_t... rest>
using aerobus::ContinuedFraction< a0, rest... >::type = q64::template add_t< typename q64::template inject_constant_t<a0>, typename q64::template div_t< typename q64::one, typename ContinuedFraction<rest...>::type > >
```

represented value as [aerobus::q64](#)

8.6.3 Member Data Documentation

8.6.3.1 val

```
template<int64_t a0, int64_t... rest>
constexpr double aerobus::ContinuedFraction< a0, rest... >::val = type::template get<double>()
[static], [constexpr]
```

represented value as double

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

- [src/aerobus.h](#)

8.7 aerobus::ConwayPolynomial Struct Reference

```
#include <aerobus.h>
```

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

- [src/aerobus.h](#)

8.8 aerobus::Embed< Small, Large, E > Struct Template Reference

embedding - struct forward declaration

8.8.1 Detailed Description

```
template<typename Small, typename Large, typename E = void>
struct aerobus::Embed< Small, Large, E >
```

embedding - struct forward declaration

Template Parameters

<i>Small</i>	a ring which can be embedded in Large
<i>Large</i>	a ring in which Small can be embedded
<i>E</i>	some default type (unused – implementation related)

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

- [src/aerobus.h](#)

8.9 aerobus::Embed< i32, i64 > Struct Reference

embeds [i32](#) into [i64](#)

```
#include <aerobus.h>
```

Public Types

- `template<typename val >`
`using type = i64::val< static_cast< int64_t >(val::v)>`
the [i64](#) representation of val

8.9.1 Detailed Description

embeds [i32](#) into [i64](#)

8.9.2 Member Typedef Documentation

8.9.2.1 type

```
template<typename val >  
using aerobus::Embed< i32, i64 >::type = i64::val<static_cast<int64_t>(val::v)>
```

the [i64](#) representation of val

Template Parameters

<i>val</i>	a value in i32
------------	--------------------------------

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

- `src/aerobus.h`

8.10 aerobus::Embed< polynomial< Small >, polynomial< Large > > Struct Template Reference

embeds `polynomial<Small>` into `polynomial<Large>`

```
#include <aerobus.h>
```

Public Types

- `template<typename v >`
`using type = typename at_low< v, typename internal::make_index_sequence_reverse< v::degree+1 > >↔`
`::type`
the `polynomial<Large>` representation of v

8.10.1 Detailed Description

```
template<typename Small, typename Large>
struct aerobus::Embed< polynomial< Small >, polynomial< Large > >
```

embeds polynomial<Small> into polynomial<Large>

Template Parameters

<i>Small</i>	a rings which can be embedded in Large
<i>Large</i>	a ring in which Small can be embedded

8.10.2 Member Typedef Documentation

8.10.2.1 type

```
template<typename Small , typename Large >
template<typename v >
using aerobus::Embed< polynomial< Small >, polynomial< Large > >::type = typename at_low<v,
typename internal::make_index_sequence_reverse<v::degree + 1> >::type
```

the polynomial<Large> reprensentation of v

Template Parameters

<i>v</i>	a value in polynomial<Small>
----------	------------------------------

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

- [src/aerobus.h](#)

8.11 aerobus::Embed< q32, q64 > Struct Reference

embeds q32 into q64

```
#include <aerobus.h>
```

Public Types

- `template<typename v >`
using `type = make_q64_t< static_cast< int64_t >(v::x::v), static_cast< int64_t >(v::y::v)>`
q64 representation of v

8.11.1 Detailed Description

embeds q32 into q64

8.11.2 Member Typedef Documentation

8.11.2.1 type

```
template<typename v >
using aerobus::Embed< q32, q64 >::type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>
```

q64 representation of v

Template Parameters

<i>v</i>	a value in q32
----------	----------------

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

- [src/aerobus.h](#)

8.12 aerobus::Embed< Quotient< Ring, X >, Ring > Struct Template Reference

embeds Quotient<Ring, X> into Ring

```
#include <aerobus.h>
```

Public Types

- `template<typename val >`
`using type = typename val::raw_t`
Ring representation of val.

8.12.1 Detailed Description

```
template<typename Ring, typename X>
struct aerobus::Embed< Quotient< Ring, X >, Ring >
```

embeds Quotient<Ring, X> into Ring

Template Parameters

<i>Ring</i>	a Euclidean ring
<i>X</i>	a value in Ring

8.12.2 Member Typedef Documentation

8.12.2.1 type

```
template<typename Ring , typename X >
template<typename val >
using aerobus::Embed< Quotient< Ring, X >, Ring >::type = typename val::raw_t
```

Ring representation of val.

Template Parameters

<i>val</i>	a value in Quotient<Ring, X>
------------	------------------------------

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

- src/[aerobus.h](#)

8.13 aerobus::Embed< Ring, FractionField< Ring > > Struct Template Reference

embeds values from Ring to its field of fractions

```
#include <aerobus.h>
```

Public Types

- template<typename v >
using [type](#) = typename [FractionField< Ring >::template val< v, typename Ring::one >](#)
FractionField<Ring> representation of v.

8.13.1 Detailed Description

```
template<typename Ring>
struct aerobus::Embed< Ring, FractionField< Ring > >
```

embeds values from Ring to its field of fractions

Template Parameters

<i>Ring</i>	an integers ring, such as i32
-------------	---

8.13.2 Member Typedef Documentation

8.13.2.1 type

```
template<typename Ring >
template<typename v >
using aerobus::Embed< Ring, FractionField< Ring > >::type = typename FractionField<Ring>←
::template val<v, typename Ring::one>
```

FractionField<Ring> representation of v.

Template Parameters

v	a Ring value
---	--------------

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

- src/[aerobus.h](#)

8.14 aerobus::Embed< zpz< x >, i32 > Struct Template Reference

embeds zpz values into [i32](#)

```
#include <aerobus.h>
```

Public Types

- `template<typename val >`
`using type = i32::val< val::v >`
the [i32](#) representation of val

8.14.1 Detailed Description

```
template<int32_t x>
struct aerobus::Embed< zpz< x >, i32 >
```

embeds zpz values into [i32](#)

Template Parameters

x	an integer
---	------------

8.14.2 Member Typedef Documentation

8.14.2.1 type

```
template<int32_t x>
template<typename val >
using aerobus::Embed< zpz< x >, i32 >::type = i32::val<val::v>
```

the `i32` representation of `val`

Template Parameters

<code>val</code>	a value in <code>zpz<x></code>
------------------	--------------------------------------

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

- `src/aerobus.h`

8.15 aerobus::i32 Struct Reference

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

```
#include <aerobus.h>
```

Classes

- struct `val`
values in `i32`, again represented as types

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`
- `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`
modulus operator yields v1 % v2 for example : `i32::mod_t<i32::val<7>, i32::val<2>>`
- `template<typename v1 , typename v2 >`
`using gt_t = typename gt< v1, v2 >::type`
- `template<typename v1 , typename v2 >`
`using lt_t = typename lt< v1, v2 >::type`
- `template<typename v1 , typename v2 >`
`using eq_t = typename eq< v1, v2 >::type`
- `template<typename v1 , typename v2 >`
`using gcd_t = gcd_t< i32, v1, v2 >`
- `template<typename v >`
`using pos_t = typename pos< v >::type`

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 eq_v = eq_t<v1, v2>::value`
- `template<typename v >`
`static constexpr bool pos_v = pos_t<v>::value`

8.15.1 Detailed Description

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

8.15.2 Member Typedef Documentation

8.15.2.1 add_t

```
template<typename v1 , typename v2 >
using aerobus::i32::add_t = typename add<v1, v2>::type
```

8.15.2.2 div_t

```
template<typename v1 , typename v2 >
using aerobus::i32::div_t = typename div<v1, v2>::type
```

8.15.2.3 eq_t

```
template<typename v1 , typename v2 >
using aerobus::i32::eq_t = typename eq<v1, v2>::type
```


8.15.2.4 gcd_t

```
template<typename v1 , typename v2 >
using aerobus::i32::gcd_t = gcd_t<i32, v1, v2>
```

8.15.2.5 gt_t

```
template<typename v1 , typename v2 >
using aerobus::i32::gt_t = typename gt<v1, v2>::type
```

8.15.2.6 inject_constant_t

```
template<auto x>
using aerobus::i32::inject_constant_t = val<static_cast<int32_t>(x)>
```

8.15.2.7 inject_ring_t

```
template<typename v >
using aerobus::i32::inject_ring_t = v
```

8.15.2.8 inner_type

```
using aerobus::i32::inner_type = int32_t
```

8.15.2.9 lt_t

```
template<typename v1 , typename v2 >
using aerobus::i32::lt_t = typename lt<v1, v2>::type
```

8.15.2.10 mod_t

```
template<typename v1 , typename v2 >
using aerobus::i32::mod_t = typename remainder<v1, v2>::type
```

modulus operator yields $v1 \% v2$ for example : `i32::mod_t<i32::val<7>, i32::val<2>>`

Template Parameters

<code>v1</code>	a value in <code>i32</code>
<code>v2</code>	a value in <code>i32</code>

8.15.2.11 mul_t

```
template<typename v1 , typename v2 >
```

```
using aerobus::i32::mul_t = typename mul<v1, v2>::type
```

8.15.2.12 one

```
using aerobus::i32::one = val<1>
```

constant one

8.15.2.13 pos_t

```
template<typename v >
using aerobus::i32::pos_t = typename pos<v>::type
```

8.15.2.14 sub_t

```
template<typename v1 , typename v2 >
using aerobus::i32::sub_t = typename sub<v1, v2>::type
```

8.15.2.15 zero

```
using aerobus::i32::zero = val<0>
```

constant zero

8.15.3 Member Data Documentation

8.15.3.1 eq_v

```
template<typename v1 , typename v2 >
constexpr bool aerobus::i32::eq_v = eq_t<v1, v2>::value [static], [constexpr]
```

8.15.3.2 is_euclidean_domain

```
constexpr bool aerobus::i32::is_euclidean_domain = true [static], [constexpr]
```

integers are an euclidean domain

8.15.3.3 is_field

```
constexpr bool aerobus::i32::is_field = false [static], [constexpr]
```

integers are not a field

8.15.3.4 pos_v

```
template<typename v >
constexpr bool aerobus::i32::pos_v = pos_t<v>::value [static], [constexpr]
```

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

- [src/aerobus.h](#)

8.16 aerobus::i64 Struct Reference

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

```
#include <aerobus.h>
```

Classes

- struct [val](#)
values in i64

Public Types

- [using inner_type = int64_t](#)
type of represented values
- [template<auto x>](#)
[using inject_constant_t = val< static_cast< int64_t >\(x\)>](#)
- [template<typename v >](#)
[using inject_ring_t = v](#)
injects a value used for internal consistency and quotient rings implementations for example i64::inject_ring_t<i64::val<1>>
-> i64::val<1>
- [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](#)
- [template<typename v1 , typename v2 >](#)
[using sub_t = typename sub< v1 , v2 >::type](#)
- [template<typename v1 , typename v2 >](#)
[using mul_t = typename mul< v1 , v2 >::type](#)
- [template<typename v1 , typename v2 >](#)
[using div_t = typename div< v1 , v2 >::type](#)
- [template<typename v1 , typename v2 >](#)
[using mod_t = typename remainder< v1 , v2 >::type](#)
- [template<typename v1 , typename v2 >](#)
[using gt_t = typename gt< v1 , v2 >::type](#)
- [template<typename v1 , typename v2 >](#)
[using lt_t = typename lt< v1 , v2 >::type](#)
- [template<typename v1 , typename v2 >](#)
[using eq_t = typename eq< v1 , v2 >::type](#)
- [template<typename v1 , typename v2 >](#)
[using gcd_t = gcd_t< i64, v1, v2 >](#)
- [template<typename v >](#)
[using pos_t = typename pos< v >::type](#)

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_t<v1, v2>::value`
strictly greater operator yields v1 > v2 as boolean value
- `template<typename v1 , typename v2 >`
`static constexpr bool lt_v = lt_t<v1, v2>::value`
- `template<typename v1 , typename v2 >`
`static constexpr bool eq_v = eq_t<v1, v2>::value`
- `template<typename v >`
`static constexpr bool pos_v = pos_t<v>::value`

8.16.1 Detailed Description

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

8.16.2 Member Typedef Documentation

8.16.2.1 add_t

```
template<typename v1 , typename v2 >
using aerobus::i64::add_t = typename add<v1, v2>::type
```

8.16.2.2 div_t

```
template<typename v1 , typename v2 >
using aerobus::i64::div_t = typename div<v1, v2>::type
```

8.16.2.3 eq_t

```
template<typename v1 , typename v2 >
using aerobus::i64::eq_t = typename eq<v1, v2>::type
```

8.16.2.4 gcd_t

```
template<typename v1 , typename v2 >
using aerobus::i64::gcd_t = gcd_t<i64, v1, v2>
```

8.16.2.5 gt_t

```
template<typename v1 , typename v2 >
using aerobus::i64::gt_t = typename gt<v1, v2>::type
```

8.16.2.6 inject_constant_t

```
template<auto x>
using aerobus::i64::inject_constant_t = val<static_cast<int64_t>(x)>
```

8.16.2.7 inject_ring_t

```
template<typename v >
using aerobus::i64::inject_ring_t = v
```

injects a value used for internal consistency and quotient rings implementations for example `i64::inject_ring_t<i64::val<1>>>`
`-> i64::val<1>`

Template Parameters

<code>v</code>	a value in <code>i64</code>
----------------	-----------------------------

8.16.2.8 inner_type

```
using aerobus::i64::inner_type = int64_t
```

type of represented values

8.16.2.9 lt_t

```
template<typename v1 , typename v2 >
using aerobus::i64::lt_t = typename lt<v1, v2>::type
```

8.16.2.10 mod_t

```
template<typename v1 , typename v2 >
using aerobus::i64::mod_t = typename remainder<v1, v2>::type
```

8.16.2.11 mul_t

```
template<typename v1 , typename v2 >
using aerobus::i64::mul_t = typename mul<v1, v2>::type
```

8.16.2.12 one

```
using aerobus::i64::one = val<1>
```

constant one

8.16.2.13 pos_t

```
template<typename v >
using aerobus::i64::pos_t = typename pos<v>::type
```

8.16.2.14 sub_t

```
template<typename v1 , typename v2 >
using aerobus::i64::sub_t = typename sub<v1, v2>::type
```

8.16.2.15 zero

```
using aerobus::i64::zero = val<0>
```

constant zero

8.16.3 Member Data Documentation

8.16.3.1 eq_v

```
template<typename v1 , typename v2 >
constexpr bool aerobus::i64::eq_v = eq_t<v1, v2>::value [static], [constexpr]
```

8.16.3.2 gt_v

```
template<typename v1 , typename v2 >
constexpr bool aerobus::i64::gt_v = gt_t<v1, v2>::value [static], [constexpr]
```

strictly greater operator yields $v1 > v2$ as boolean value

Template Parameters

<code>v1</code>	: an element of aerobus::i64::val
<code>v2</code>	: an element of aerobus::i64::val

8.16.3.3 is_euclidean_domain

```
constexpr bool aerobus::i64::is_euclidean_domain = true [static], [constexpr]
```

integers are an euclidean domain

8.16.3.4 is_field

```
constexpr bool aerobus::i64::is_field = false [static], [constexpr]
```

integers are not a field

8.16.3.5 lt_v

```
template<typename v1 , typename v2 >
constexpr bool aerobus::i64::lt_v = lt_t<v1, v2>::value [static], [constexpr]
```

8.16.3.6 pos_v

```
template<typename v >
constexpr bool aerobus::i64::pos_v = pos_t<v>::value [static], [constexpr]
```

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

- [src/aerobus.h](#)

8.17 aerobus::is_prime< n > Struct Template Reference

checks if n is prime

```
#include <aerobus.h>
```

Static Public Attributes

- static constexpr bool [value](#) = internal::_is_prime<n, 5>::value
true iff n is prime

8.17.1 Detailed Description

```
template<size_t n>
struct aerobus::is_prime< n >
```

checks if n is prime

Template Parameters

<i>n</i>	
----------	--

8.17.2 Member Data Documentation

8.17.2.1 value

```
template<size_t n>
constexpr bool aerobus::is_prime< n >::value = internal::_is_prime<n, 5>::value [static],
[constexpr]
```

true iff n is prime

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

- [src/aerobus.h](#)

8.18 aerobus::polynomial< Ring > Struct Template Reference

```
#include <aerobus.h>
```

Classes

- struct [val](#)
values (seen as types) in polynomial ring
- struct [val< coeffN >](#)
specialization for constants

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 (recursively deletes highest degree if zero, 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](#)
subtraction of two polynomials
- [template<typename v1 , typename v2 >](#)
[using mul_t = typename mul< v1, v2 >::type](#)
multiplication of two polynomials
- [template<typename v1 , typename v2 >](#)
[using eq_t = typename eq_helper< v1, v2 >::type](#)
equality operator
- [template<typename v1 , typename v2 >](#)
[using lt_t = typename lt_helper< v1, v2 >::type](#)
strict less operator
- [template<typename v1 , typename v2 >](#)
[using gt_t = typename gt_helper< v1, v2 >::type](#)
strict greater operator
- [template<typename v1 , typename v2 >](#)
[using div_t = typename div< v1, v2 >::q_type](#)
division operator

- `template<typename v1 , typename v2 >`
`using mod_t = typename div_helper< v1, v2, zero, v1 >::mod_type`
modulo operator
- `template<typename coeff , size_t deg>`
`using monomial_t = typename monomial< coeff, deg >::type`
monomial : coeff X^deg
- `template<typename v >`
`using derive_t = typename derive_helper< v >::type`
derivation operator
- `template<typename v >`
`using pos_t = typename Ring::template pos_t< typename v::aN >`
checks for positivity (an > 0)
- `template<typename v1 , typename v2 >`
`using gcd_t = std::conditional_t< Ring::is_euclidean_domain, typename make_unit< gcd_t< polynomial<`
`Ring >, v1, v2 > >::type, void >`
greatest common divisor of two polynomials
- `template<auto x>`
`using inject_constant_t = val< typename Ring::template inject_constant_t< x > >`
- `template<typename v >`
`using inject_ring_t = val< v >`

Static Public Attributes

- `static constexpr bool is_field = false`
- `static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain`
- `template<typename v >`
`static constexpr bool pos_v = pos_t<v>::value`
positivity operator

8.18.1 Detailed Description

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

polynomial with coefficients in Ring Ring must be an integral domain

8.18.2 Member Typedef Documentation

8.18.2.1 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

<i>v1</i>	
<i>v2</i>	

8.18.2.2 derive_t

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

derivation operator

Template Parameters

<i>v</i>	
----------	--

8.18.2.3 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

<i>v1</i>	
<i>v2</i>	

8.18.2.4 eq_t

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

equality operator

Template Parameters

<i>v1</i>	
<i>v2</i>	

8.18.2.5 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

<i>v1</i>	
<i>v2</i>	

8.18.2.6 gt_t

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

strict greater operator

Template Parameters

<i>v1</i>	
<i>v2</i>	

8.18.2.7 inject_constant_t

```
template<typename Ring >
template<auto x>
using aerobus::polynomial< Ring >::inject_constant_t = val<typename Ring::template inject_constant_t<x>
>
```

8.18.2.8 inject_ring_t

```
template<typename Ring >
template<typename v >
using aerobus::polynomial< Ring >::inject_ring_t = val<v>
```

8.18.2.9 lt_t

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

strict less operator

Template Parameters

<i>v1</i>	
<i>v2</i>	

8.18.2.10 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

<i>v1</i>	
<i>v2</i>	

8.18.2.11 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

<i>coeff</i>	
<i>deg</i>	

8.18.2.12 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

<i>v1</i>	
<i>v2</i>	

8.18.2.13 one

```
template<typename Ring >
using aerobus::polynomial< Ring >::one = val<typename Ring::one>
```

constant one

8.18.2.14 pos_t

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

checks for positivity (an > 0)

Template Parameters

<i>v</i>	
----------	--

8.18.2.15 simplify_t

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

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

Template Parameters

<i>P</i>	
----------	--

8.18.2.16 sub_t

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

subtraction of two polynomials

Template Parameters

<i>v1</i>	
<i>v2</i>	

8.18.2.17 X

```
template<typename Ring >
using aerobus::polynomial< Ring >::X = val<typename Ring::one, typename Ring::zero>
```

generator

8.18.2.18 zero

```
template<typename Ring >
using aerobus::polynomial< Ring >::zero = val<typename Ring::zero>
```

constant zero

8.18.3 Member Data Documentation

8.18.3.1 is_euclidean_domain

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

8.18.3.2 is_field

```
template<typename Ring >
constexpr bool aerobus::polynomial< Ring >::is_field = false [static], [constexpr]
```

8.18.3.3 pos_v

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

positivity operator

Template Parameters

<i>v</i>	a value in polynomial::val
----------	--

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

- [src/aerobus.h](#)

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

removes types from head of the list

```
#include <aerobus.h>
```

Public Types

- using [type](#) = typename internal::pop_front_h< Ts... >::head
type that was previously head of the list
- using [tail](#) = typename internal::pop_front_h< Ts... >::tail
remaining types in parent list when front is removed

8.19.1 Detailed Description

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

removes types from head of the list

8.19.2 Member Typedef Documentation

8.19.2.1 tail

```
template<typename... Ts>
using aerobus::type_list< Ts >::pop_front::tail = typename internal::pop_front_h<Ts...>::tail
```

remaining types in parent list when front is removed

8.19.2.2 type

```
template<typename... Ts>
using aerobus::type_list< Ts >::pop_front::type = typename internal::pop_front_h<Ts...>::head
```

type that was previously head of the list

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

- [src/aerobus.h](#)

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

[Quotient](#) ring by the principal ideal generated by 'X' With [i32](#) as Ring and [i32::val<2>](#) as X, [Quotient](#) is $\mathbb{Z}/2\mathbb{Z}$.

```
#include <aerobus.h>
```

Classes

- struct [val](#)
projection values in the quotient ring

Public Types

- using [zero](#) = [val](#)< [typename](#) [Ring](#)::zero >
zero value
- using [one](#) = [val](#)< [typename](#) [Ring](#)::one >
one
- template<[typename](#) [v1](#) , [typename](#) [v2](#) >
using [add_t](#) = [val](#)< [typename](#) [Ring](#)::template [add_t](#)< [typename](#) [v1](#)::type, [typename](#) [v2](#)::type > >
addition operator
- template<[typename](#) [v1](#) , [typename](#) [v2](#) >
using [mul_t](#) = [val](#)< [typename](#) [Ring](#)::template [mul_t](#)< [typename](#) [v1](#)::type, [typename](#) [v2](#)::type > >
subtraction operator
- template<[typename](#) [v1](#) , [typename](#) [v2](#) >
using [div_t](#) = [val](#)< [typename](#) [Ring](#)::template [div_t](#)< [typename](#) [v1](#)::type, [typename](#) [v2](#)::type > >
division operator
- template<[typename](#) [v1](#) , [typename](#) [v2](#) >
using [mod_t](#) = [val](#)< [typename](#) [Ring](#)::template [mod_t](#)< [typename](#) [v1](#)::type, [typename](#) [v2](#)::type > >

- modulus operator*
 • `template<typename v1 , typename v2 >`
 `using eq_t = typename Ring::template eq_t< typename v1::type, typename v2::type >`
 equality operator (as type)
- `template<typename v1 >`
 `using pos_t = std::true_type`
 positivity operator always true
- `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_t<typename v1::type, typename v2::type>::value`
 addition operator (as boolean value)
- `template<typename v >`
 `static constexpr bool pos_v = pos_t<v>::value`
 positivity operator always true
- `static constexpr bool is_euclidean_domain = true`
 quotien rings are euclidean domain

8.20.1 Detailed Description

`template<typename Ring, typename X>`
requires `IsRing<Ring>`
struct `aerobus::Quotient< Ring, X >`

`Quotient` ring by the principal ideal generated by 'X' With `i32` as Ring and `i32::val<2>` as X, `Quotient` is $\mathbb{Z}/2\mathbb{Z}$.

Template Parameters

<i>Ring</i>	A ring type, such as ' <code>i32</code> ', must satisfy the <code>IsRing</code> concept
<i>X</i>	a value in Ring, such as <code>i32::val<2></code>

8.20.2 Member Typedef Documentation

8.20.2.1 add_t

`template<typename Ring , typename X >`
`template<typename v1 , typename v2 >`
`using aerobus::Quotient< Ring, X >::add_t = val<typename Ring::template add_t<typename v1::type,`
`typename v2::type> >`

addition operator

Template Parameters

<i>v1</i>	a value in quotient ring
<i>v2</i>	a value in quotient ring

8.20.2.2 div_t

```
template<typename Ring , typename X >
template<typename v1 , typename v2 >
using aerobus::Quotient< Ring, X >::div_t = val<typename Ring::template div_t<typename v1::type,
typename v2::type> >
```

division operator

Template Parameters

<i>v1</i>	a value in quotient ring
<i>v2</i>	a value in quotient ring

8.20.2.3 eq_t

```
template<typename Ring , typename X >
template<typename v1 , typename v2 >
using aerobus::Quotient< Ring, X >::eq_t = typename Ring::template eq_t<typename v1::type,
typename v2::type>
```

equality operator (as type)

Template Parameters

<i>v1</i>	a value in quotient ring
<i>v2</i>	a value in quotient ring

8.20.2.4 inject_constant_t

```
template<typename Ring , typename X >
template<auto x>
using aerobus::Quotient< Ring, X >::inject_constant_t = val<typename Ring::template inject_constant_t<x>
>
```

8.20.2.5 inject_ring_t

```
template<typename Ring , typename X >
template<typename v >
using aerobus::Quotient< Ring, X >::inject_ring_t = val<v>
```

8.20.2.6 mod_t

```
template<typename Ring , typename X >
template<typename v1 , typename v2 >
using aerobus::Quotient< Ring, X >::mod_t = val<typename Ring::template mod_t<typename v1::type,
typename v2::type> >
```

modulus operator

Template Parameters

v1	a value in quotient ring
v2	a value in quotient ring

8.20.2.7 mul_t

```
template<typename Ring , typename X >
template<typename v1 , typename v2 >
using aerobus::Quotient< Ring, X >::mul_t = val<typename Ring::template mul_t<typename v1::type,
typename v2::type> >
```

subtraction operator

Template Parameters

v1	a value in quotient ring
v2	a value in quotient ring

8.20.2.8 one

```
template<typename Ring , typename X >
using aerobus::Quotient< Ring, X >::one = val<typename Ring::one>
```

one

8.20.2.9 pos_t

```
template<typename Ring , typename X >
template<typename v1 >
using aerobus::Quotient< Ring, X >::pos_t = std::true_type
```

positivity operator always true

Template Parameters

v1	a value in quotient ring
----	--------------------------

8.20.2.10 zero

```
template<typename Ring , typename X >
using aerobus::Quotient< Ring, X >::zero = val<typename Ring::zero>
```

zero value

8.20.3 Member Data Documentation

8.20.3.1 eq_v

```
template<typename Ring , typename X >
template<typename v1 , typename v2 >
constexpr bool aerobus::Quotient< Ring, X >::eq_v = Ring::template eq_t<typename v1::type,
typename v2::type>::value [static], [constexpr]
```

addition operator (as boolean value)

Template Parameters

<i>v1</i>	a value in quotient ring
<i>v2</i>	a value in quotient ring

8.20.3.2 is_euclidean_domain

```
template<typename Ring , typename X >
constexpr bool aerobus::Quotient< Ring, X >::is_euclidean_domain = true [static], [constexpr]
```

quotien rings are euclidean domain

8.20.3.3 pos_v

```
template<typename Ring , typename X >
template<typename v >
constexpr bool aerobus::Quotient< Ring, X >::pos_v = pos_t<v>::value [static], [constexpr]
```

positivity operator always true

Template Parameters

<i>v1</i>	a value in quotient ring
-----------	--------------------------

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

- [src/aerobus.h](#)

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

splits list at index

```
#include <aerobus.h>
```

Public Types

- using [head](#) = typename inner::head
- using [tail](#) = typename inner::tail

8.21.1 Detailed Description

```
template<typename... Ts>
template<size_t index>
struct aerobus::type_list< Ts >::split< index >
```

splits list at index

Template Parameters

<i>index</i>	
--------------	--

8.21.2 Member Typedef Documentation

8.21.2.1 head

```
template<typename... Ts>
template<size_t index>
using aerobus::type_list< Ts >::split< index >::head = typename inner::head
```

8.21.2.2 tail

```
template<typename... Ts>
template<size_t index>
using aerobus::type_list< Ts >::split< index >::tail = typename inner::tail
```

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

- src/[aerobus.h](#)

8.22 aerobus::type_list< Ts > Struct Template Reference

Empty pure template struct to handle type list.

```
#include <aerobus.h>
```

Classes

- struct [pop_front](#)
removes types from head of the list
- struct [split](#)
splits list at index

Public Types

- template<typename T >
using [push_front](#) = [type_list](#)< T, Ts... >
Adds T to front of the list.
- template<size_t index>
using [at](#) = [internal::type_at_t](#)< index, Ts... >
returns type at index
- template<typename T >
using [push_back](#) = [type_list](#)< Ts..., T >
pushes T at the tail of the list
- template<typename U >
using [concat](#) = typename [concat_h](#)< U >::type
concatenates two list into one
- template<typename T , size_t index>
using [insert](#) = typename [internal::insert_h](#)< index, [type_list](#)< Ts... >, T >::type
inserts type at index
- template<size_t index>
using [remove](#) = typename [internal::remove_h](#)< index, [type_list](#)< Ts... > >::type
removes type at index

Static Public Attributes

- static constexpr size_t [length](#) = sizeof...(Ts)
length of list

8.22.1 Detailed Description

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

Empty pure template struct to handle type list.

8.22.2 Member Typedef Documentation

8.22.2.1 at

```
template<typename... Ts>
template<size_t index>
using aerobus::type_list< Ts >::at = internal::type\_at\_t<index, Ts...>
```

returns type at index

Template Parameters

<i>index</i>	
--------------	--

8.22.2.2 concat

```
template<typename... Ts>
template<typename U >
using aerobus::type_list< Ts >::concat = typename concat_h<U>::type
```

concatenates two list into one

Template Parameters

<i>U</i>	
----------	--

8.22.2.3 insert

```
template<typename... Ts>
template<typename T , size_t index>
using aerobus::type_list< Ts >::insert = typename internal::insert_h<index, type_list<Ts...>,
T>::type
```

inserts type at index

Template Parameters

<i>index</i>	
<i>T</i>	

8.22.2.4 push_back

```
template<typename... Ts>
template<typename T >
using aerobus::type_list< Ts >::push_back = type_list<Ts..., T>
```

pushes T at the tail of the list

Template Parameters

<i>T</i>	
----------	--

8.22.2.5 push_front

```
template<typename... Ts>
```

```
template<typename T >
using aerobus::type_list< Ts >::push_front = type_list<T, Ts...>
```

Adds T to front of the list.

Template Parameters

<i>T</i>	
----------	--

8.22.2.6 remove

```
template<typename... Ts>
template<size_t index>
using aerobus::type_list< Ts >::remove = typename internal::remove_h<index, type_list<Ts...>
>::type
```

removes type at index

Template Parameters

<i>index</i>	
--------------	--

8.22.3 Member Data Documentation

8.22.3.1 length

```
template<typename... Ts>
constexpr size_t aerobus::type_list< Ts >::length = sizeof...(Ts) [static], [constexpr]
```

length of list

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

- src/[aerobus.h](#)

8.23 aerobus::type_list<> Struct Reference

specialization for empty type list

```
#include <aerobus.h>
```

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<typename T , size_t index>
using [insert](#) = [type_list](#)< T >

Static Public Attributes

- static constexpr size_t `length` = 0

8.23.1 Detailed Description

specialization for empty type list

8.23.2 Member Typedef Documentation

8.23.2.1 `concat`

```
template<typename U >
using aerobus::type_list<>::concat = U
```

8.23.2.2 `insert`

```
template<typename T , size_t index>
using aerobus::type_list<>::insert = type_list<T>
```

8.23.2.3 `push_back`

```
template<typename T >
using aerobus::type_list<>::push_back = type_list<T>
```

8.23.2.4 `push_front`

```
template<typename T >
using aerobus::type_list<>::push_front = type_list<T>
```

8.23.3 Member Data Documentation

8.23.3.1 `length`

```
constexpr size_t aerobus::type_list<>::length = 0 [static], [constexpr]
```

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

- src/[aerobus.h](#)

8.24 `aerobus::i32::val< x >` Struct Template Reference

values in [i32](#), again represented as types

```
#include <aerobus.h>
```


Public Types

- `using enclosing_type = i32`
Enclosing ring type.
- `using is_zero_t = std::bool_constant< x==0 >`
is value zero

Static Public Member Functions

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

Static Public Attributes

- `static constexpr int32_t v = x`
actual value stored in val type

8.24.1 Detailed Description

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

values in `i32`, again represented as types

Template Parameters

<code>x</code>	an actual integer
----------------	-------------------

8.24.2 Member Typedef Documentation

8.24.2.1 enclosing_type

```
template<int32_t x>
using aerobus::i32::val< x >::enclosing_type = i32
```

Enclosing ring type.

8.24.2.2 is_zero_t

```
template<int32_t x>
using aerobus::i32::val< x >::is_zero_t = std::bool_constant<x == 0>
```

is value zero

8.24.3 Member Function Documentation

8.24.3.1 eval()

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

cast x into valueRing

Template Parameters

<i>valueRing</i>	double for example
------------------	--------------------

8.24.3.2 get()

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

cast x into valueType

Template Parameters

<i>valueType</i>	double for example
------------------	--------------------

8.24.3.3 to_string()

```
template<int32_t x>
static std::string aerobus::i32::val< x >::to_string ( ) [inline], [static]
```

string representation of value

8.24.4 Member Data Documentation

8.24.4.1 v

```
template<int32_t x>
constexpr int32_t aerobus::i32::val< x >::v = x [static], [constexpr]
```

actual value stored in val type

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

- [src/aerobus.h](#)

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

values in [i64](#)

```
#include <aerobus.h>
```

Public Types

- [using inner_type = int32_t](#)
type of represented values
- [using enclosing_type = i64](#)
enclosing ring type
- [using is_zero_t = std::bool_constant< x==0 >](#)
is value zero

Static Public Member Functions

- [template<typename valueType >](#)
[static constexpr DEVICE INLINED valueType get \(\)](#)
cast value in valueType
- [static std::string to_string \(\)](#)
string representation
- [template<typename valueRing >](#)
[static constexpr DEVICE INLINED valueRing eval \(const valueRing &v\)](#)
cast value in valueRing

Static Public Attributes

- [static constexpr int64_t v = x](#)
actual value

8.25.1 Detailed Description

```
template<int64\_t x>
struct aerobus::i64::val< x >
```

values in [i64](#)

Template Parameters

x	an actual integer
-------------------	-------------------

8.25.2 Member Typedef Documentation

8.25.2.1 enclosing_type

```
template<int64_t x>
using aerobus::i64::val< x >::enclosing_type = i64
```

enclosing ring type

8.25.2.2 inner_type

```
template<int64_t x>
using aerobus::i64::val< x >::inner_type = int32_t
```

type of represented values

8.25.2.3 is_zero_t

```
template<int64_t x>
using aerobus::i64::val< x >::is_zero_t = std::bool_constant<x == 0>
```

is value zero

8.25.3 Member Function Documentation

8.25.3.1 eval()

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

cast value in valueRing

Template Parameters

<i>valueRing</i>	(double for example)
------------------	----------------------

8.25.3.2 get()

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

cast value in valueType

Template Parameters

<i>valueType</i>	(double for example)
------------------	----------------------

8.25.3.3 to_string()

```
template<int64_t x>
static std::string aerobus::i64::val< x >::to_string ( ) [inline], [static]
```

string representation

8.25.4 Member Data Documentation

8.25.4.1 v

```
template<int64_t x>
constexpr int64_t aerobus::i64::val< x >::v = x [static], [constexpr]
```

actual value

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

- [src/aerobus.h](#)

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

values (seen as types) in polynomial ring

```
#include <aerobus.h>
```

Public Types

- [using ring_type = Ring](#)
ring coefficients live in
- [using enclosing_type = polynomial< Ring >](#)
enclosing ring type
- [using aN = coeffN](#)
heavy weight coefficient (non zero)
- [using strip = val< coeffs... >](#)
remove largest coefficient
- [using is_zero_t = std::bool_constant<\(degree==0\) &&\(aN::is_zero_t::value\)>](#)
true_type if polynomial is constant zero
- [template<size_t index>](#)
[using coeff_at_t = typename coeff_at< index >::type](#)
type of coefficient at index

Static Public Member Functions

- `static std::string to_string ()`
get a string representation of polynomial
- `template<typename valueRing >`
`static constexpr DEVICE INLINED 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 = is_zero_t::value`
true if polynomial is constant zero

8.26.1 Detailed Description

```
template<typename Ring>
template<typename coeffN, typename... coeffs>
struct aerobus::polynomial< Ring >::val< coeffN, coeffs >
```

values (seen as types) in polynomial ring

Template Parameters

<code>coeffN</code>	high degree coefficient
<code>...coeffs</code>	lower degree coefficients

8.26.2 Member Typedef Documentation

8.26.2.1 aN

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
using aerobus::polynomial< Ring >::val< coeffN, coeffs >::aN = coeffN
```

heavy weight coefficient (non zero)

8.26.2.2 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
```

type of coefficient at index

Template Parameters

<i>index</i>	
--------------	--

8.26.2.3 enclosing_type

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
using aerobus::polynomial< Ring >::val< coeffN, coeffs >::enclosing_type = polynomial<Ring>
```

enclosing ring type

8.26.2.4 is_zero_t

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
using aerobus::polynomial< Ring >::val< coeffN, coeffs >::is_zero_t = std::bool_constant<(degree
== 0) && (aN::is_zero_t::value)>
```

true_type if polynomial is constant zero

8.26.2.5 ring_type

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
using aerobus::polynomial< Ring >::val< coeffN, coeffs >::ring_type = Ring
```

ring coefficients live in

8.26.2.6 strip

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
using aerobus::polynomial< Ring >::val< coeffN, coeffs >::strip = val<coeffs...>
```

remove largest coefficient

8.26.3 Member Function Documentation**8.26.3.1 eval()**

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
template<typename valueRing >
static constexpr DEVICE INLINED 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

<i>valueRing</i>	usually float or double
------------------	-------------------------

Parameters

<i>x</i>	value
----------	-------

Returns

$P(x)$

8.26.3.2 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 $a_n X^n + \dots + a_1 X + a_0$

8.26.4 Member Data Documentation

8.26.4.1 degree

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
constexpr size_t aerobus::polynomial< Ring >::val< coeffN, coeffs >::degree = sizeof...(coeffs)
[static], [constexpr]
```

degree of the polynomial

8.26.4.2 is_zero_v

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
constexpr bool aerobus::polynomial< Ring >::val< coeffN, coeffs >::is_zero_v = is_zero_t←
::value [static], [constexpr]
```

true if polynomial is constant zero

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

- [src/aerobus.h](#)

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

projection values in the quotient ring

```
#include <aerobus.h>
```

Public Types

- `using raw_t = V`
- `using type = abs_t< typename Ring::template mod_t< V, X > >`

8.27.1 Detailed Description

```
template<typename Ring, typename X>
template<typename V>
struct aerobus::Quotient< Ring, X >::val< V >
```

projection values in the quotient ring

Template Parameters

V	a value from 'Ring'
---	---------------------

8.27.2 Member Typedef Documentation

8.27.2.1 raw_t

```
template<typename Ring , typename X >
template<typename V >
using aerobus::Quotient< Ring, X >::val< V >::raw_t = V
```

8.27.2.2 type

```
template<typename Ring , typename X >
template<typename V >
using aerobus::Quotient< Ring, X >::val< V >::type = abs_t<typename Ring::template mod_t<V,
X> >
```

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

- `src/aerobus.h`

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

values in zpz

```
#include <aerobus.h>
```

Public Types

- `using enclosing_type = zpz< p >`
enclosing ring type
- `using is_zero_t = std::bool_constant< v==0 >`
true_type if zero

Static Public Member Functions

- `template<typename valueType >`
`static constexpr DEVICE INLINED valueType get ()`
get value as valueType
- `static std::string to_string ()`
string representation
- `template<typename valueRing >`
`static constexpr DEVICE INLINED valueRing eval (const valueRing &v)`

Static Public Attributes

- `static constexpr int32_t v = x % p`
actual value
- `static constexpr bool is_zero_v = v == 0`
true if zero

8.28.1 Detailed Description

```
template<int32_t p>
template<int32_t x>
struct aerobus::zpz< p >::val< x >
```

values in zpz

Template Parameters

<i>x</i>	an integer
----------	------------

8.28.2 Member Typedef Documentation

8.28.2.1 enclosing_type

```
template<int32_t p>
template<int32_t x>
using aerobus::zpz< p >::val< x >::enclosing_type = zpz<p>
```

enclosing ring type

8.28.2.2 is_zero_t

```
template<int32_t p>
template<int32_t x>
using aerobus::zpz< p >::val< x >::is_zero_t = std::bool_constant<v == 0>
```

true_type if zero

8.28.3 Member Function Documentation

8.28.3.1 eval()

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

8.28.3.2 get()

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

get value as valueType

Template Parameters

<i>valueType</i>	an arithmetic type, such as float
------------------	-----------------------------------

8.28.3.3 to_string()

```
template<int32_t p>
template<int32_t x>
static std::string aerobus::zpz< p >::val< x >::to_string ( ) [inline], [static]
```

string representation

Returns

a string representation

8.28.4 Member Data Documentation

8.28.4.1 is_zero_v

```
template<int32_t p>
template<int32_t x>
constexpr bool aerobus::zpz< p >::val< x >::is_zero_v = v == 0 [static], [constexpr]
```

true if zero

8.28.4.2 v

```
template<int32_t p>
template<int32_t x>
constexpr int32_t aerobus::zpz< p >::val< x >::v = x % p [static], [constexpr]
```

actual value

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

- src/aerobus.h

8.29 aerobus::polynomial< Ring >::val< coeffN > Struct Template Reference

specialization for constants

```
#include <aerobus.h>
```

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 ring_type = Ring](#)
ring coefficients live in
- [using enclosing_type = polynomial< Ring >](#)
enclosing ring type
- [using aN = coeffN](#)
- [using strip = val< coeffN >](#)
- [using is_zero_t = std::bool_constant< aN::is_zero_t::value >](#)
- [template<size_t index>](#)
[using coeff_at_t = typename coeff_at< index >::type](#)

Static Public Member Functions

- [static std::string to_string \(\)](#)
- [template<typename valueRing >](#)
[static constexpr DEVICE INLINED valueRing eval \(const valueRing &x\)](#)

Static Public Attributes

- [static constexpr size_t degree = 0](#)
degree
- [static constexpr bool is_zero_v = is_zero_t::value](#)

8.29.1 Detailed Description

```
template<typename Ring>
template<typename coeffN>
struct aerobus::polynomial< Ring >::val< coeffN >
```

specialization for constants

Template Parameters

<i>coeffN</i>	
---------------	--

8.29.2 Member Typedef Documentation

8.29.2.1 aN

```
template<typename Ring >
template<typename coeffN >
using aerobus::polynomial< Ring >::val< coeffN >::aN = coeffN
```

8.29.2.2 coeff_at_t

```
template<typename Ring >
template<typename coeffN >
template<size_t index>
using aerobus::polynomial< Ring >::val< coeffN >::coeff_at_t = typename coeff_at<index>↵
::type
```

8.29.2.3 enclosing_type

```
template<typename Ring >
template<typename coeffN >
using aerobus::polynomial< Ring >::val< coeffN >::enclosing_type = polynomial<Ring>
```

enclosing ring type

8.29.2.4 is_zero_t

```
template<typename Ring >
template<typename coeffN >
using aerobus::polynomial< Ring >::val< coeffN >::is_zero_t = std::bool_constant<aN::is_↵
zero_t::value>
```

8.29.2.5 ring_type

```
template<typename Ring >
template<typename coeffN >
using aerobus::polynomial< Ring >::val< coeffN >::ring_type = Ring
```

ring coefficients live in

8.29.2.6 strip

```
template<typename Ring >
template<typename coeffN >
using aerobus::polynomial< Ring >::val< coeffN >::strip = val<coeffN>
```

8.29.3 Member Function Documentation

8.29.3.1 eval()

```
template<typename Ring >
template<typename coeffN >
template<typename valueRing >
static constexpr DEVICE INLINED valueRing aerobus::polynomial< Ring >::val< coeffN >::eval (
    const valueRing & x ) [inline], [static], [constexpr]
```

8.29.3.2 to_string()

```
template<typename Ring >
template<typename coeffN >
static std::string aerobus::polynomial< Ring >::val< coeffN >::to_string ( ) [inline], [static]
```

8.29.4 Member Data Documentation

8.29.4.1 degree

```
template<typename Ring >
template<typename coeffN >
constexpr size_t aerobus::polynomial< Ring >::val< coeffN >::degree = 0 [static], [constexpr]
```

degree

8.29.4.2 is_zero_v

```
template<typename Ring >
template<typename coeffN >
constexpr bool aerobus::polynomial< Ring >::val< coeffN >::is_zero_v = is_zero_t::value [static],
[constexpr]
```

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

- [src/aerobus.h](#)

8.30 aerobus::zpz< p > Struct Template Reference

congruence classes of integers modulo p (32 bits)

```
#include <aerobus.h>
```

Classes

- struct [val](#)
values in zpz

Public Types

- `using inner_type = int32_t`
underlying type for values
- `template<auto x>`
`using inject_constant_t = val< static_cast< int32_t >(x)>`
injects a constant integer into zpz
- `using zero = val< 0 >`
zero value
- `using one = val< 1 >`
one value
- `template<typename v1 , typename v2 >`
`using add_t = typename add< v1, v2 >::type`
addition operator
- `template<typename v1 , typename v2 >`
`using sub_t = typename sub< v1, v2 >::type`
subtraction 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`
modulo operator
- `template<typename v1 , typename v2 >`
`using gt_t = typename gt< v1, v2 >::type`
strictly greater operator (type)
- `template<typename v1 , typename v2 >`
`using lt_t = typename lt< v1, v2 >::type`
strictly smaller operator (type)
- `template<typename v1 , typename v2 >`
`using eq_t = typename eq< v1, v2 >::type`
equality operator (type)
- `template<typename v1 , typename v2 >`
`using gcd_t = gcd_t< i32, v1, v2 >`
greatest common divisor
- `template<typename v1 >`
`using pos_t = typename pos< v1 >::type`
positivity operator (type)

Static Public Attributes

- `static constexpr bool is_field = is_prime<p>::value`
true iff p is prime
- `static constexpr bool is_euclidean_domain = true`
always true
- `template<typename v1 , typename v2 >`
`static constexpr bool gt_v = gt_t<v1, v2>::value`
strictly greater operator (booleanvalue)

- `template<typename v1 , typename v2 >`
`static constexpr bool lt_v = lt_t<v1, v2>::value`
strictly smaller operator (booleanvalue)
- `template<typename v1 , typename v2 >`
`static constexpr bool eq_v = eq_t<v1, v2>::value`
equality operator (booleanvalue)
- `template<typename v >`
`static constexpr bool pos_v = pos_t<v>::value`
positivity operator (boolean value)

8.30.1 Detailed Description

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

congruence classes of integers modulo p (32 bits)

if p is prime, zpz

is a field

Template Parameters

<i>p</i>	a integer
----------	-----------

8.30.2 Member Typedef Documentation

8.30.2.1 add_t

```
template<int32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz< p >::add_t = typename add<v1, v2>::type
```

addition operator

Template Parameters

<i>v1</i>	a value in zpz::val
<i>v2</i>	a value in zpz::val

8.30.2.2 div_t

```
template<int32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz< p >::div_t = typename div<v1, v2>::type
```

division operator

Template Parameters

<i>v1</i>	a value in zpz::val
<i>v2</i>	a value in zpz::val

8.30.2.3 eq_t

```
template<int32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz< p >::eq_t = typename eq<v1, v2>::type
```

equality operator (type)

Template Parameters

<i>v1</i>	a value in zpz::val
<i>v2</i>	a value in zpz::val

8.30.2.4 gcd_t

```
template<int32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz< p >::gcd_t = gcd_t<i32, v1, v2>
```

greatest common divisor

Template Parameters

<i>v1</i>	a value in zpz::val
<i>v2</i>	a value in zpz::val

8.30.2.5 gt_t

```
template<int32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz< p >::gt_t = typename gt<v1, v2>::type
```

strictly greater operator (type)

Template Parameters

<i>v1</i>	a value in zpz::val
<i>v2</i>	a value in zpz::val

8.30.2.6 inject_constant_t

```
template<int32_t p>
template<auto x>
using aerobus::zpz< p >::inject_constant_t = val<static_cast<int32_t>(x)>
```

injects a constant integer into zpz

Template Parameters

<i>x</i>	an integer
----------	------------

8.30.2.7 inner_type

```
template<int32_t p>
using aerobus::zpz< p >::inner_type = int32_t
```

underlying type for values

8.30.2.8 lt_t

```
template<int32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz< p >::lt_t = typename lt<v1, v2>::type
```

strictly smaller operator (type)

Template Parameters

<i>v1</i>	a value in zpz::val
<i>v2</i>	a value in zpz::val

8.30.2.9 mod_t

```
template<int32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz< p >::mod_t = typename remainder<v1, v2>::type
```

modulo operator

Template Parameters

<i>v1</i>	a value in zpz::val
<i>v2</i>	a value in zpz::val

8.30.2.10 mul_t

```
template<int32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz< p >::mul_t = typename mul<v1, v2>::type
```

multiplication operator

Template Parameters

v1	a value in zpz::val
v2	a value in zpz::val

8.30.2.11 one

```
template<int32_t p>
using aerobus::zpz< p >::one = val<1>
```

one value

8.30.2.12 pos_t

```
template<int32_t p>
template<typename v1 >
using aerobus::zpz< p >::pos_t = typename pos<v1>::type
```

positivity operator (type)

Template Parameters

v1	a value in zpz::val
----	-------------------------------------

8.30.2.13 sub_t

```
template<int32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz< p >::sub_t = typename sub<v1, v2>::type
```

subtraction operator

Template Parameters

v1	a value in zpz::val
v2	a value in zpz::val

8.30.2.14 zero

```
template<int32_t p>
using aerobus::zpz< p >::zero = val<0>
```

zero value

8.30.3 Member Data Documentation

8.30.3.1 eq_v

```
template<int32_t p>
template<typename v1 , typename v2 >
constexpr bool aerobus::zpz< p >::eq_v = eq_t<v1, v2>::value [static], [constexpr]
```

equality operator (booleanvalue)

Template Parameters

v1	a value in zpz::val
v2	a value in zpz::val

8.30.3.2 gt_v

```
template<int32_t p>
template<typename v1 , typename v2 >
constexpr bool aerobus::zpz< p >::gt_v = gt_t<v1, v2>::value [static], [constexpr]
```

strictly greater operator (booleanvalue)

Template Parameters

v1	a value in zpz::val
v2	a value in zpz::val

8.30.3.3 is_euclidean_domain

```
template<int32_t p>
constexpr bool aerobus::zpz< p >::is_euclidean_domain = true [static], [constexpr]
```

always true

8.30.3.4 is_field

```
template<int32_t p>
constexpr bool aerobus::zpz< p >::is_field = is_prime<p>::value [static], [constexpr]
```

true iff p is prime

8.30.3.5 lt_v

```
template<int32_t p>
template<typename v1 , typename v2 >
constexpr bool aerobus::zpz< p >::lt_v = lt_t<v1, v2>::value [static], [constexpr]
```

strictly smaller operator (boolean value)

Template Parameters

<i>v1</i>	a value in zpz::val
<i>v2</i>	a value in zpz::val

8.30.3.6 pos_v

```
template<int32_t p>
template<typename v >
constexpr bool aerobus::zpz< p >::pos_v = pos_t<v>::value [static], [constexpr]
```

positivity operator (boolean value)

Template Parameters

<i>v1</i>	a value in zpz::val
-----------	-------------------------------------

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

- [src/aerobus.h](#)

Chapter 9

File Documentation

9.1 README.md File Reference

9.2 src/aerobus.h File Reference

```
#include <cstdint>
#include <cstddef>
#include <cstring>
#include <type_traits>
#include <utility>
#include <algorithm>
#include <functional>
#include <string>
#include <concepts>
#include <array>
```

Include dependency graph for aerobus.h:

9.3 aerobus.h

[Go to the documentation of this file.](#)

```
00001 // -*- lsst-c++ -*-
00002 #ifndef __INC_AEROBUS__ // NOLINT
00003 #define __INC_AEROBUS__
00004
00005 #include <cstdint>
00006 #include <cstddef>
00007 #include <cstring>
00008 #include <type_traits>
00009 #include <utility>
00010 #include <algorithm>
00011 #include <functional>
00012 #include <string>
00013 #include <concepts> // NOLINT
00014 #include <array>
00015
00019 #ifdef _MSC_VER
00020 #define ALIGNED(x) __declspec(align(x))
00021 #define INLINED __forceinline
00022 #else
00023 #define ALIGNED(x) __attribute__((aligned(x)))
00024 #define INLINED __attribute__((always_inline)) inline
00025 #endif
00026
00027 #ifdef __CUDACC__
00028 #define DEVICE __host__ __device__
```

```

00029 #else
00030 #define DEVICE
00031 #endif
00032
00033
00034
00035
00036
00037
00038
00039 // aligned allocation
00040 namespace aerobus {
00041     template<typename T>
00042     T* aligned_malloc(size_t count, size_t alignment) {
00043         #ifdef _MSC_VER
00044             return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
00045         #else
00046             return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
00047         #endif
00048     }
00049 } // namespace aerobus
00050
00051 // concepts
00052 namespace aerobus {
00053     template <typename R>
00054     concept IsRing = requires {
00055         typename R::one;
00056         typename R::zero;
00057         typename R::template add_t<typename R::one, typename R::one>;
00058         typename R::template sub_t<typename R::one, typename R::one>;
00059         typename R::template mul_t<typename R::one, typename R::one>;
00060     };
00061
00062     template <typename R>
00063     concept IsEuclideanDomain = IsRing<R> && requires {
00064         typename R::template div_t<typename R::one, typename R::one>;
00065         typename R::template mod_t<typename R::one, typename R::one>;
00066         typename R::template gcd_t<typename R::one, typename R::one>;
00067         typename R::template eq_t<typename R::one, typename R::one>;
00068         typename R::template pos_t<typename R::one>;
00069
00070         R::template pos_v<typename R::one> == true;
00071         // typename R::template gt_t<typename R::one, typename R::zero>;
00072         R::is_euclidean_domain == true;
00073     };
00074
00075     template<typename R>
00076     concept IsField = IsEuclideanDomain<R> && requires {
00077         R::is_field == true;
00078     };
00079 } // namespace aerobus
00080
00081 // utilities
00082 namespace aerobus {
00083     namespace internal {
00084         template<template<typename...> typename TT, typename T>
00085         struct is_instantiation_of : std::false_type { };
00086
00087         template<template<typename...> typename TT, typename... Ts>
00088         struct is_instantiation_of<TT, TT<Ts...> : std::true_type { };
00089
00090         template<template<typename...> typename TT, typename T>
00091         inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00092
00093         template <int64_t i, typename T, typename... Ts>
00094         struct type_at {
00095             static_assert(i < sizeof...(Ts) + 1, "index out of range");
00096             using type = typename type_at<i - 1, Ts...>::type;
00097         };
00098
00099         template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00100             using type = T;
00101         };
00102
00103         template <size_t i, typename... Ts>
00104         using type_at_t = typename type_at<i, Ts...>::type;
00105
00106         template<size_t n, size_t i, typename E = void>
00107         struct _is_prime {};
00108
00109         template<size_t i>
00110         struct _is_prime<0, i> {
00111             static constexpr bool value = false;
00112         };
00113
00114         template<size_t i>
00115         struct _is_prime<1, i> {
00116             static constexpr bool value = false;
00117         };
00118     }
00119 }

```



```

00128
00129     template<size_t i>
00130     struct _is_prime<2, i> {
00131         static constexpr bool value = true;
00132     };
00133
00134     template<size_t i>
00135     struct _is_prime<3, i> {
00136         static constexpr bool value = true;
00137     };
00138
00139     template<size_t i>
00140     struct _is_prime<5, i> {
00141         static constexpr bool value = true;
00142     };
00143
00144     template<size_t i>
00145     struct _is_prime<7, i> {
00146         static constexpr bool value = true;
00147     };
00148
00149     template<size_t n, size_t i>
00150     struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)>> {
00151         static constexpr bool value = false;
00152     };
00153
00154     template<size_t n, size_t i>
00155     struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)>> {
00156         static constexpr bool value = false;
00157     };
00158
00159     template<size_t n, size_t i>
00160     struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)>> {
00161         static constexpr bool value = true;
00162     };
00163
00164     template<size_t n, size_t i>
00165     struct _is_prime<n, i, std::enable_if_t<(
00166         n % i == 0 &&
00167         n >= 9 &&
00168         n % 3 != 0 &&
00169         n % 2 != 0 &&
00170         i * i > n)>> {
00171         static constexpr bool value = true;
00172     };
00173
00174     template<size_t n, size_t i>
00175     struct _is_prime<n, i, std::enable_if_t<(
00176         n % (i+2) == 0 &&
00177         n >= 9 &&
00178         n % 3 != 0 &&
00179         n % 2 != 0 &&
00180         i * i <= n)>> {
00181         static constexpr bool value = true;
00182     };
00183
00184     template<size_t n, size_t i>
00185     struct _is_prime<n, i, std::enable_if_t<(
00186         n % (i+2) != 0 &&
00187         n % i != 0 &&
00188         n >= 9 &&
00189         n % 3 != 0 &&
00190         n % 2 != 0 &&
00191         (i * i <= n))>> {
00192         static constexpr bool value = _is_prime<n, i+6>::value;
00193     };
00194
00195 } // namespace internal
00196
00197 template<size_t n>
00198 struct is_prime {
00199     static constexpr bool value = internal::_is_prime<n, 5>::value;
00200 };
00201
00202 template<size_t n>
00203 static constexpr bool is_prime_v = is_prime<n>::value;
00204
00205 // gcd
00206 namespace internal {
00207     template <std::size_t... Is>
00208     constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
00209         -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
00210
00211     template <std::size_t N>
00212     using make_index_sequence_reverse
00213         = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00214
00215 }
00216
00217
00218
00219
00220

```

```

00226     template<typename Ring, typename E = void>
00227     struct gcd;
00228
00229     template<typename Ring>
00230     struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain> {
00231         template<typename A, typename B, typename E = void>
00232         struct gcd_helper {};
00233
00234         // B = 0, A > 0
00235         template<typename A, typename B>
00236         struct gcd_helper<A, B, std::enable_if_t<
00237             (B::is_zero_t::value) &&
00238             (Ring::template gt_t<A, typename Ring::zero>::value)>> {
00239             using type = A;
00240         };
00241
00242         // B = 0, A < 0
00243         template<typename A, typename B>
00244         struct gcd_helper<A, B, std::enable_if_t<
00245             (B::is_zero_t::value) &&
00246             !(Ring::template gt_t<A, typename Ring::zero>::value)>> {
00247             using type = typename Ring::template sub_t<typename Ring::zero, A>;
00248         };
00249
00250         // B != 0
00251         template<typename A, typename B>
00252         struct gcd_helper<A, B, std::enable_if_t<
00253             (!B::is_zero_t::value)
00254             >> {
00255             private: // NOLINT
00256                 // A / B
00257                 using k = typename Ring::template div_t<A, B>;
00258                 // A - (A/B)*B = A % B
00259                 using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B>;
00260
00261             public:
00262                 using type = typename gcd_helper<B, m>::type;
00263         };
00264
00265         template<typename A, typename B>
00266         using type = typename gcd_helper<A, B>::type;
00267     };
00268 } // namespace internal
00269
00270 // vadd and vmul
00271 namespace internal {
00272     template<typename... vals>
00273     struct vmul {};
00274
00275     template<typename v1, typename... vals>
00276     struct vmul<v1, vals...> {
00277         using type = typename v1::enclosing_type::template mul_t<v1, typename
vmul<vals...>::type>;
00278     };
00279
00280     template<typename v1>
00281     struct vmul<v1> {
00282         using type = v1;
00283     };
00284
00285     template<typename... vals>
00286     struct vadd {};
00287
00288     template<typename v1, typename... vals>
00289     struct vadd<v1, vals...> {
00290         using type = typename v1::enclosing_type::template add_t<v1, typename
vadd<vals...>::type>;
00291     };
00292
00293     template<typename v1>
00294     struct vadd<v1> {
00295         using type = v1;
00296     };
00297 } // namespace internal
00298
00299 template<typename T, typename A, typename B>
00300 using gcd_t = typename internal::gcd<T>::template type<A, B>;
00301
00302 template<typename... vals>
00303 using vadd_t = typename internal::vadd<vals...>::type;
00304
00305 template<typename... vals>
00306 using vmul_t = typename internal::vmul<vals...>::type;
00307
00308 template<typename val>
00309 requires IsEuclideanDomain<typename val::enclosing_type>
00310 using abs_t = std::conditional_t<

```

```

00322         val::enclosing_type::template pos_v<val>,
00323         val, typename val::enclosing_type::template sub_t<typename
    val::enclosing_type::zero, val>;
00324 } // namespace aerobus
00325
00326 // embedding
00327 namespace aerobus {
00332     template<typename Small, typename Large, typename E = void>
00333     struct Embed;
00334 } // namespace aerobus
00335
00336 namespace aerobus {
00341     template<typename Ring, typename X>
00342     requires IsRing<Ring>
00343     struct Quotient {
00346         template <typename V>
00347         struct val {
00348             public:
00349                 using raw_t = V;
00350                 using type = abs_t<typename Ring::template mod_t<V, X>>;
00351         };
00352
00354         using zero = val<typename Ring::zero>;
00355
00357         using one = val<typename Ring::one>;
00358
00362         template<typename v1, typename v2>
00363         using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00364
00368         template<typename v1, typename v2>
00369         using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00370
00374         template<typename v1, typename v2>
00375         using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00376
00380         template<typename v1, typename v2>
00381         using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00382
00386         template<typename v1, typename v2>
00387         using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00388
00392         template<typename v1, typename v2>
00393         static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00394
00398         template<typename v1>
00399         using pos_t = std::true_type;
00400
00404         template<typename v>
00405         static constexpr bool pos_v = pos_t<v>::value;
00406
00408         static constexpr bool is_euclidean_domain = true;
00409
00415         template<auto x>
00416         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00417
00423         template<typename v>
00424         using inject_ring_t = val<v>;
00425     };
00426
00430     template<typename Ring, typename X>
00431     struct Embed<Quotient<Ring, X>, Ring> {
00434         template<typename val>
00435         using type = typename val::raw_t;
00436     };
00437 } // namespace aerobus
00438
00439 // type_list
00440 namespace aerobus {
00442     template <typename... Ts>
00443     struct type_list;
00444
00445     namespace internal {
00446         template <typename T, typename... Us>
00447         struct pop_front_h {
00448             using tail = type_list<Us...>;
00449             using head = T;
00450         };
00451
00452         template <size_t index, typename L1, typename L2>
00453         struct split_h {
00454             private:
00455                 static_assert(index <= L2::length, "index out of bounds");
00456                 using a = typename L2::pop_front::type;
00457                 using b = typename L2::pop_front::tail;
00458                 using c = typename L1::template push_back<a>;
00459
00460             public:

```

```

00461         using head = typename split_h<index - 1, c, b>::head;
00462         using tail = typename split_h<index - 1, c, b>::tail;
00463     };
00464
00465     template <typename L1, typename L2>
00466     struct split_h<0, L1, L2> {
00467         using head = L1;
00468         using tail = L2;
00469     };
00470
00471     template <size_t index, typename L, typename T>
00472     struct insert_h {
00473         static_assert(index <= L::length, "index out of bounds");
00474         using s = typename L::template split<index>;
00475         using left = typename s::head;
00476         using right = typename s::tail;
00477         using ll = typename left::template push_back<T>;
00478         using type = typename ll::template concat<right>;
00479     };
00480
00481     template <size_t index, typename L>
00482     struct remove_h {
00483         using s = typename L::template split<index>;
00484         using left = typename s::head;
00485         using right = typename s::tail;
00486         using rr = typename right::pop_front::tail;
00487         using type = typename left::template concat<rr>;
00488     };
00489 } // namespace internal
00490
00491 template <typename... Ts>
00492 struct type_list {
00493 private:
00494     template <typename T>
00495     struct concat_h;
00496
00497     template <typename... Us>
00498     struct concat_h<type_list<Us...> {
00499         using type = type_list<Ts..., Us...>;
00500     };
00501
00502 public:
00503     static constexpr size_t length = sizeof...(Ts);
00504
00505     template <typename T>
00506     using push_front = type_list<T, Ts...>;
00507
00508     template <size_t index>
00509     using at = internal::type_at_t<index, Ts...>;
00510
00511     struct pop_front {
00512         using type = typename internal::pop_front_h<Ts...>::head;
00513         using tail = typename internal::pop_front_h<Ts...>::tail;
00514     };
00515
00516     template <typename T>
00517     using push_back = type_list<Ts..., T>;
00518
00519     template <typename U>
00520     using concat = typename concat_h<U>::type;
00521
00522     template <size_t index>
00523     struct split {
00524 private:
00525         using inner = internal::split_h<index, type_list<>, type_list<Ts...>>;
00526
00527     public:
00528         using head = typename inner::head;
00529         using tail = typename inner::tail;
00530     };
00531
00532     template <typename T, size_t index>
00533     using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00534
00535     template <size_t index>
00536     using remove = typename internal::remove_h<index, type_list<Ts...>::type;
00537 };
00538
00539 template <>
00540 struct type_list<> {
00541     static constexpr size_t length = 0;
00542
00543     template <typename T>
00544     using push_front = type_list<T>;
00545
00546     template <typename T>
00547     using push_back = type_list<T>;

```

```

00571
00572     template <typename U>
00573     using concat = U;
00574
00575     // TODO(jewave): assert index == 0
00576     template <typename T, size_t index>
00577     using insert = type_list<T>;
00578 };
00579 } // namespace aerobus
00580
00581 // i32
00582 namespace aerobus {
00583     struct i32 {
00584         using inner_type = int32_t;
00585         template<int32_t x>
00586         struct val {
00587             using enclosing_type = i32;
00588             static constexpr int32_t v = x;
00589
00590             template<typename valueType>
00591             static constexpr INLINED DEVICE valueType get() { return static_cast<valueType>(x); }
00592
00593             using is_zero_t = std::bool_constant<x == 0>;
00594
00595             static std::string to_string() {
00596                 return std::to_string(x);
00597             }
00598
00599             template<typename valueRing>
00600             static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
00601                 return static_cast<valueRing>(x);
00602             }
00603         };
00604     };
00605
00606     using zero = val<0>;
00607     using one = val<1>;
00608     static constexpr bool is_field = false;
00609     static constexpr bool is_euclidean_domain = true;
00610     template<auto x>
00611     using inject_constant_t = val<static_cast<int32_t>(x)>;
00612
00613     template<typename v>
00614     using inject_ring_t = v;
00615
00616 private:
00617     template<typename v1, typename v2>
00618     struct add {
00619         using type = val<v1::v + v2::v>;
00620     };
00621
00622     template<typename v1, typename v2>
00623     struct sub {
00624         using type = val<v1::v - v2::v>;
00625     };
00626
00627     template<typename v1, typename v2>
00628     struct mul {
00629         using type = val<v1::v * v2::v>;
00630     };
00631
00632     template<typename v1, typename v2>
00633     struct div {
00634         using type = val<v1::v / v2::v>;
00635     };
00636
00637     template<typename v1, typename v2>
00638     struct remainder {
00639         using type = val<v1::v % v2::v>;
00640     };
00641
00642     template<typename v1, typename v2>
00643     struct gt {
00644         using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00645     };
00646
00647     template<typename v1, typename v2>
00648     struct lt {
00649         using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00650     };
00651
00652     template<typename v1, typename v2>
00653     struct eq {
00654         using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00655     };
00656
00657     template<typename v1>
00658     struct pos {

```

```

00676         using type = std::bool_constant<(v1::v > 0)>;
00677     };
00678
00679     public:
00685         template<typename v1, typename v2>
00686         using add_t = typename add<v1, v2>::type;
00687
00693         template<typename v1, typename v2>
00694         using sub_t = typename sub<v1, v2>::type;
00695
00701         template<typename v1, typename v2>
00702         using mul_t = typename mul<v1, v2>::type;
00703
00709         template<typename v1, typename v2>
00710         using div_t = typename div<v1, v2>::type;
00711
00717         template<typename v1, typename v2>
00718         using mod_t = typename remainder<v1, v2>::type;
00719
00725         template<typename v1, typename v2>
00726         using gt_t = typename gt<v1, v2>::type;
00727
00733         template<typename v1, typename v2>
00734         using lt_t = typename lt<v1, v2>::type;
00735
00741         template<typename v1, typename v2>
00742         using eq_t = typename eq<v1, v2>::type;
00743
00748         template<typename v1, typename v2>
00749         static constexpr bool eq_v = eq_t<v1, v2>::value;
00750
00756         template<typename v1, typename v2>
00757         using gcd_t = gcd_t<i32, v1, v2>;
00758
00763         template<typename v>
00764         using pos_t = typename pos<v>::type;
00765
00770         template<typename v>
00771         static constexpr bool pos_v = pos_t<v>::value;
00772     };
00773 } // namespace aerobus
00774
00775 // i64
00776 namespace aerobus {
00777     struct i64 {
00780         using inner_type = int64_t;
00781         template<int64_t x>
00782         struct val {
00786             using inner_type = int32_t;
00787             using enclosing_type = i64;
00788             static constexpr int64_t v = x;
00791
00794             template<typename valueType>
00795             static constexpr DEVICE INLINE valueType get() {
00796                 return static_cast<valueType>(x);
00797             }
00798
00800             using is_zero_t = std::bool_constant<x == 0>;
00801
00803             static std::string to_string() {
00804                 return std::to_string(x);
00805             }
00806
00809             template<typename valueRing>
00810             static constexpr DEVICE INLINE valueRing eval(const valueRing& v) {
00811                 return static_cast<valueRing>(x);
00812             }
00813         };
00814
00818         template<auto x>
00819         using inject_constant_t = val<static_cast<int64_t>(x)>;
00820
00825         template<typename v>
00826         using inject_ring_t = v;
00827
00829         using zero = val<0>;
00831         using one = val<1>;
00833         static constexpr bool is_field = false;
00835         static constexpr bool is_euclidean_domain = true;
00836
00837     private:
00838         template<typename v1, typename v2>
00839         struct add {
00840             using type = val<v1::v + v2::v>;
00841         };
00842
00843         template<typename v1, typename v2>

```

```

00844     struct sub {
00845         using type = val<v1::v - v2::v>;
00846     };
00847
00848     template<typename v1, typename v2>
00849     struct mul {
00850         using type = val<v1::v* v2::v>;
00851     };
00852
00853     template<typename v1, typename v2>
00854     struct div {
00855         using type = val<v1::v / v2::v>;
00856     };
00857
00858     template<typename v1, typename v2>
00859     struct remainder {
00860         using type = val<v1::v% v2::v>;
00861     };
00862
00863     template<typename v1, typename v2>
00864     struct gt {
00865         using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00866     };
00867
00868     template<typename v1, typename v2>
00869     struct lt {
00870         using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00871     };
00872
00873     template<typename v1, typename v2>
00874     struct eq {
00875         using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00876     };
00877
00878     template<typename v>
00879     struct pos {
00880         using type = std::bool_constant<(v::v > 0)>;
00881     };
00882
00883 public:
00884     template<typename v1, typename v2>
00885     using add_t = typename add<v1, v2>::type;
00886
00887     template<typename v1, typename v2>
00888     using sub_t = typename sub<v1, v2>::type;
00889
00890     template<typename v1, typename v2>
00891     using mul_t = typename mul<v1, v2>::type;
00892
00893     template<typename v1, typename v2>
00894     using div_t = typename div<v1, v2>::type;
00895
00896     template<typename v1, typename v2>
00897     using mod_t = typename remainder<v1, v2>::type;
00898
00899     template<typename v1, typename v2>
00900     using gt_t = typename gt<v1, v2>::type;
00901
00902     template<typename v1, typename v2>
00903     static constexpr bool gt_v = gt_t<v1, v2>::value;
00904
00905     template<typename v1, typename v2>
00906     using lt_t = typename lt<v1, v2>::type;
00907
00908     template<typename v1, typename v2>
00909     static constexpr bool lt_v = lt_t<v1, v2>::value;
00910
00911     template<typename v1, typename v2>
00912     using eq_t = typename eq<v1, v2>::type;
00913
00914     template<typename v1, typename v2>
00915     static constexpr bool eq_v = eq_t<v1, v2>::value;
00916
00917     template<typename v1, typename v2>
00918     using gcd_t = gcd_t<i64, v1, v2>;
00919
00920     template<typename v>
00921     using pos_t = typename pos<v>::type;
00922
00923     template<typename v>
00924     static constexpr bool pos_v = pos_t<v>::value;
00925 };
00926
00927 template<>
00928 struct Embed<i32, i64> {
00929     template<typename val>
00930     using type = i64::val<static_cast<int64_t>(val::v)>;

```

```

00997     };
00998 } // namespace aerobus
00999
01000 // z/pz
01001 namespace aerobus {
01002     template<int32_t p>
01003     struct zpz {
01004         using inner_type = int32_t;
01005
01006         template<int32_t x>
01007         struct val {
01008             using enclosing_type = zpz<p>;
01009             static constexpr int32_t v = x % p;
01010
01011             template<typename valueType>
01012             static constexpr DEVICE INLINED valueType get() { return static_cast<valueType>(x % p); }
01013
01014             using is_zero_t = std::bool_constant<v == 0>;
01015
01016             static constexpr bool is_zero_v = v == 0;
01017
01018             static std::string to_string() {
01019                 return std::to_string(x % p);
01020             }
01021
01022             template<typename valueRing>
01023             static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
01024                 return static_cast<valueRing>(x % p);
01025             }
01026         };
01027     };
01028
01029     template<auto x>
01030     using inject_constant_t = val<static_cast<int32_t>(x)>;
01031
01032     using zero = val<0>;
01033
01034     using one = val<1>;
01035
01036     static constexpr bool is_field = is_prime<p>::value;
01037
01038     static constexpr bool is_euclidean_domain = true;
01039
01040 private:
01041     template<typename v1, typename v2>
01042     struct add {
01043         using type = val<(v1::v + v2::v) % p>;
01044     };
01045
01046     template<typename v1, typename v2>
01047     struct sub {
01048         using type = val<(v1::v - v2::v) % p>;
01049     };
01050
01051     template<typename v1, typename v2>
01052     struct mul {
01053         using type = val<(v1::v * v2::v) % p>;
01054     };
01055
01056     template<typename v1, typename v2>
01057     struct div {
01058         using type = val<(v1::v % p) / (v2::v % p)>;
01059     };
01060
01061     template<typename v1, typename v2>
01062     struct remainder {
01063         using type = val<(v1::v % v2::v) % p>;
01064     };
01065
01066     template<typename v1, typename v2>
01067     struct gt {
01068         using type = std::conditional_t<(v1::v % p > v2::v % p), std::true_type, std::false_type>;
01069     };
01070
01071     template<typename v1, typename v2>
01072     struct lt {
01073         using type = std::conditional_t<(v1::v % p < v2::v % p), std::true_type, std::false_type>;
01074     };
01075
01076     template<typename v1, typename v2>
01077     struct eq {
01078         using type = std::conditional_t<(v1::v % p == v2::v % p), std::true_type, std::false_type>;
01079     };
01080
01081     template<typename v1>
01082     struct pos {
01083         using type = std::bool_constant<(v1::v > 0)>;
01084     };
01085 }
01086

```



```

01106
01107     public:
01111         template<typename v1, typename v2>
01112             using add_t = typename add<v1, v2>::type;
01113
01117         template<typename v1, typename v2>
01118             using sub_t = typename sub<v1, v2>::type;
01119
01123         template<typename v1, typename v2>
01124             using mul_t = typename mul<v1, v2>::type;
01125
01129         template<typename v1, typename v2>
01130             using div_t = typename div<v1, v2>::type;
01131
01135         template<typename v1, typename v2>
01136             using mod_t = typename remainder<v1, v2>::type;
01137
01141         template<typename v1, typename v2>
01142             using gt_t = typename gt<v1, v2>::type;
01143
01147         template<typename v1, typename v2>
01148             static constexpr bool gt_v = gt_t<v1, v2>::value;
01149
01153         template<typename v1, typename v2>
01154             using lt_t = typename lt<v1, v2>::type;
01155
01159         template<typename v1, typename v2>
01160             static constexpr bool lt_v = lt_t<v1, v2>::value;
01161
01165         template<typename v1, typename v2>
01166             using eq_t = typename eq<v1, v2>::type;
01167
01171         template<typename v1, typename v2>
01172             static constexpr bool eq_v = eq_t<v1, v2>::value;
01173
01177         template<typename v1, typename v2>
01178             using gcd_t = gcd_t<i32, v1, v2>;
01179
01182         template<typename v1>
01183             using pos_t = typename pos<v1>::type;
01184
01187         template<typename v>
01188             static constexpr bool pos_v = pos_t<v>::value;
01189     };
01190
01193     template<int32_t x>
01194     struct Embed<zp<x>, i32> {
01197         template <typename val>
01198             using type = i32::val<val::v>;
01199     };
01200 } // namespace aerobus
01201
01202 // polynomial
01203 namespace aerobus {
01204     // coeffN x^N + ...
01209     template<typename Ring>
01210     requires IsEuclideanDomain<Ring>
01211     struct polynomial {
01212         static constexpr bool is_field = false;
01213         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01214
01218         template<typename coeffN, typename... coeffs>
01219         struct val {
01221             using ring_type = Ring;
01222             using enclosing_type = polynomial<Ring>;
01225             static constexpr size_t degree = sizeof...(coeffs);
01227             using aN = coeffN;
01229             using strip = val<coeffs...>;
01231             using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01233             static constexpr bool is_zero_v = is_zero_t::value;
01234
01235         private:
01236             template<size_t index, typename E = void>
01237                 struct coeff_at {};
01238
01239             template<size_t index>
01240             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))> {
01241                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01242             };
01243
01244             template<size_t index>
01245             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))> {
01246                 using type = typename Ring::zero;
01247             };
01248
01249         public:
01252             template<size_t index>

```

```

01253         using coeff_at_t = typename coeff_at<index>::type;
01254
01257     static std::string to_string() {
01258         return string_helper<coeffN, coeffs...>::func();
01259     }
01260
01265     template<typename valueRing>
01266     static constexpr DEVICE INLINED valueRing eval(const valueRing& x) {
01267         return horner_evaluation<valueRing, val>
01268             ::template inner<0, degree + 1>
01269             ::func(static_cast<valueRing>(0), x);
01270     }
01271 };
01272
01275 template<typename coeffN>
01276 struct val<coeffN> {
01277     using ring_type = Ring;
01278     using enclosing_type = polynomial<Ring>;
01282     static constexpr size_t degree = 0;
01283     using aN = coeffN;
01284     using strip = val<coeffN>;
01285     using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01286
01287     static constexpr bool is_zero_v = is_zero_t::value;
01288
01289     template<size_t index, typename E = void>
01290     struct coeff_at {};
01291
01292     template<size_t index>
01293     struct coeff_at<index, std::enable_if_t<(index == 0)>> {
01294         using type = aN;
01295     };
01296
01297     template<size_t index>
01298     struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)>> {
01299         using type = typename Ring::zero;
01300     };
01301
01302     template<size_t index>
01303     using coeff_at_t = typename coeff_at<index>::type;
01304
01305     static std::string to_string() {
01306         return string_helper<coeffN>::func();
01307     }
01308
01309     template<typename valueRing>
01310     static constexpr DEVICE INLINED valueRing eval(const valueRing& x) {
01311         return static_cast<valueRing>(aN::template get<valueRing>());
01312     }
01313 };
01314
01316 using zero = val<typename Ring::zero>;
01318 using one = val<typename Ring::one>;
01320 using X = val<typename Ring::one, typename Ring::zero>;
01321
01322 private:
01323     template<typename P, typename E = void>
01324     struct simplify;
01325
01326     template <typename P1, typename P2, typename I>
01327     struct add_low;
01328
01329     template<typename P1, typename P2>
01330     struct add {
01331         using type = typename simplify<typename add_low<
01332             P1,
01333             P2,
01334             internal::make_index_sequence_reverse<
01335                 std::max(P1::degree, P2::degree) + 1
01336             >::type>::type;
01337     };
01338
01339     template <typename P1, typename P2, typename I>
01340     struct sub_low;
01341
01342     template <typename P1, typename P2, typename I>
01343     struct mul_low;
01344
01345     template<typename v1, typename v2>
01346     struct mul {
01347         using type = typename mul_low<
01348             v1,
01349             v2,
01350             internal::make_index_sequence_reverse<
01351                 v1::degree + v2::degree + 1
01352             >::type>;
01353     };

```

```

01354
01355     template<typename coeff, size_t deg>
01356     struct monomial;
01357
01358     template<typename v, typename E = void>
01359     struct derive_helper {};
01360
01361     template<typename v>
01362     struct derive_helper<v, std::enable_if_t<v::degree == 0> {
01363         using type = zero;
01364     };
01365
01366     template<typename v>
01367     struct derive_helper<v, std::enable_if_t<v::degree != 0> {
01368         using type = typename add<
01369             typename derive_helper<typename simplify<typename v::strip>::type>::type,
01370             typename monomial<
01371                 typename Ring::template mul_t<
01372                     typename v::aN,
01373                     typename Ring::template inject_constant_t<(v::degree)>
01374                 >,
01375                 v::degree - 1
01376             >::type
01377         >::type;
01378     };
01379
01380     template<typename v1, typename v2, typename E = void>
01381     struct eq_helper {};
01382
01383     template<typename v1, typename v2>
01384     struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree> {
01385         using type = std::false_type;
01386     };
01387
01388     template<typename v1, typename v2>
01389     struct eq_helper<v1, v2, std::enable_if_t<
01390         v1::degree == v2::degree &&
01391         (v1::degree != 0 || v2::degree != 0) &&
01392         std::is_same<
01393             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01394             std::false_type
01395         >::value
01396     > {
01397     > {
01398         using type = std::false_type;
01399     };
01400
01401     template<typename v1, typename v2>
01402     struct eq_helper<v1, v2, std::enable_if_t<
01403         v1::degree == v2::degree &&
01404         (v1::degree != 0 || v2::degree != 0) &&
01405         std::is_same<
01406             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01407             std::true_type
01408         >::value
01409     >> {
01410         using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01411     };
01412
01413     template<typename v1, typename v2>
01414     struct eq_helper<v1, v2, std::enable_if_t<
01415         v1::degree == v2::degree &&
01416         (v1::degree == 0)
01417     >> {
01418         using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01419     };
01420
01421     template<typename v1, typename v2, typename E = void>
01422     struct lt_helper {};
01423
01424     template<typename v1, typename v2>
01425     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)> {
01426         using type = std::true_type;
01427     };
01428
01429     template<typename v1, typename v2>
01430     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)> {
01431         using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01432     };
01433
01434     template<typename v1, typename v2>
01435     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)> {
01436         using type = std::false_type;
01437     };
01438
01439     template<typename v1, typename v2, typename E = void>
01440

```

```

01441     struct gt_helper {};
01442
01443     template<typename v1, typename v2>
01444     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)>> {
01445         using type = std::true_type;
01446     };
01447
01448     template<typename v1, typename v2>
01449     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)>> {
01450         using type = std::false_type;
01451     };
01452
01453     template<typename v1, typename v2>
01454     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)>> {
01455         using type = std::false_type;
01456     };
01457
01458     // when high power is zero : strip
01459     template<typename P>
01460     struct simplify<P, std::enable_if_t<
01461         std::is_same<
01462             typename Ring::zero,
01463             typename P::aN
01464         >::value && (P::degree > 0)
01465     >> {
01466         using type = typename simplify<typename P::strip>::type;
01467     };
01468
01469     // otherwise : do nothing
01470     template<typename P>
01471     struct simplify<P, std::enable_if_t<
01472         !std::is_same<
01473             typename Ring::zero,
01474             typename P::aN
01475         >::value && (P::degree > 0)
01476     >> {
01477         using type = P;
01478     };
01479
01480     // do not simplify constants
01481     template<typename P>
01482     struct simplify<P, std::enable_if_t<P::degree == 0>> {
01483         using type = P;
01484     };
01485
01486     // addition at
01487     template<typename P1, typename P2, size_t index>
01488     struct add_at {
01489         using type =
01490             typename Ring::template add_t<
01491                 typename P1::template coeff_at_t<index>,
01492                 typename P2::template coeff_at_t<index>>;
01493     };
01494
01495     template<typename P1, typename P2, size_t index>
01496     using add_at_t = typename add_at<P1, P2, index>::type;
01497
01498     template<typename P1, typename P2, std::size_t... I>
01499     struct add_low<P1, P2, std::index_sequence<I...>> {
01500         using type = val<add_at_t<P1, P2, I>...>;
01501     };
01502
01503     // subtraction at
01504     template<typename P1, typename P2, size_t index>
01505     struct sub_at {
01506         using type =
01507             typename Ring::template sub_t<
01508                 typename P1::template coeff_at_t<index>,
01509                 typename P2::template coeff_at_t<index>>;
01510     };
01511
01512     template<typename P1, typename P2, size_t index>
01513     using sub_at_t = typename sub_at<P1, P2, index>::type;
01514
01515     template<typename P1, typename P2, std::size_t... I>
01516     struct sub_low<P1, P2, std::index_sequence<I...>> {
01517         using type = val<sub_at_t<P1, P2, I>...>;
01518     };
01519
01520     template<typename P1, typename P2>
01521     struct sub {
01522         using type = typename simplify<typename sub_low<
01523             P1,
01524             P2,
01525             internal::make_index_sequence<
01526                 std::max(P1::degree, P2::degree) + 1
01527             >::type>::type;

```

```

01528     };
01529
01530     // multiplication at
01531     template<typename v1, typename v2, size_t k, size_t index, size_t stop>
01532     struct mul_at_loop_helper {
01533         using type = typename Ring::template add_t<
01534             typename Ring::template mul_t<
01535                 typename v1::template coeff_at_t<index>,
01536                 typename v2::template coeff_at_t<k - index>
01537             >,
01538             typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
01539         >;
01540     };
01541
01542     template<typename v1, typename v2, size_t k, size_t stop>
01543     struct mul_at_loop_helper<v1, v2, k, stop, stop> {
01544         using type = typename Ring::template mul_t<
01545             typename v1::template coeff_at_t<stop>,
01546             typename v2::template coeff_at_t<0>>;
01547     };
01548
01549     template<typename v1, typename v2, size_t k, typename E = void>
01550     struct mul_at {};
01551
01552     template<typename v1, typename v2, size_t k>
01553     struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)>> {
01554         using type = typename Ring::zero;
01555     };
01556
01557     template<typename v1, typename v2, size_t k>
01558     struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)>> {
01559         using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01560     };
01561
01562     template<typename P1, typename P2, size_t index>
01563     using mul_at_t = typename mul_at<P1, P2, index>::type;
01564
01565     template<typename P1, typename P2, std::size_t... I>
01566     struct mul_low<P1, P2, std::index_sequence<I...> {
01567         using type = val<mul_at_t<P1, P2, I>...>;
01568     };
01569
01570     // division helper
01571     template<typename A, typename B, typename Q, typename R, typename E = void>
01572     struct div_helper {};
01573
01574     template<typename A, typename B, typename Q, typename R>
01575     struct div_helper<A, B, Q, R, std::enable_if_t<
01576         (R::degree < B::degree) ||
01577         (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)>> {
01578         using q_type = Q;
01579         using mod_type = R;
01580         using gcd_type = B;
01581     };
01582
01583     template<typename A, typename B, typename Q, typename R>
01584     struct div_helper<A, B, Q, R, std::enable_if_t<
01585         (R::degree >= B::degree) &&
01586         !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)>> {
01587     private: // NOLINT
01588         using rN = typename R::aN;
01589         using bN = typename B::aN;
01590         using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
01591             B::degree>::type;
01592         using rr = typename sub<R, typename mul<pT, B>::type>::type;
01593         using qq = typename add<Q, pT>::type;
01594
01595     public:
01596         using q_type = typename div_helper<A, B, qq, rr>::q_type;
01597         using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
01598         using gcd_type = rr;
01599     };
01600
01601     template<typename A, typename B>
01602     struct div {
01603         static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
01604         using q_type = typename div_helper<A, B, zero, A>::q_type;
01605         using m_type = typename div_helper<A, B, zero, A>::mod_type;
01606     };
01607
01608     template<typename P>
01609     struct make_unit {
01610         using type = typename div<P, val<typename P::aN>>::q_type;
01611     };
01612
01613     template<typename coeff, size_t deg>
01614     struct monomial {

```

```

01614         using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
01615     };
01616
01617     template<typename coeff>
01618     struct monomial<coeff, 0> {
01619         using type = val<coeff>;
01620     };
01621
01622     template<typename valueRing, typename P>
01623     struct horner_evaluation {
01624         template<size_t index, size_t stop>
01625         struct inner {
01626             static constexpr DEVICE INLINED valueRing func(const valueRing& accum, const
valueRing& x) {
01627                 constexpr valueRing coeff =
01628                     static_cast<valueRing>(P::template coeff_at_t<P::degree - index>::template
get<valueRing>());
01629                 return horner_evaluation<valueRing, P>::template inner<index + 1, stop>::func(x *
accum + coeff, x);
01630             }
01631         };
01632
01633         template<size_t stop>
01634         struct inner<stop, stop> {
01635             static constexpr DEVICE INLINED valueRing func(const valueRing& accum, const
valueRing& x) {
01636                 return accum;
01637             }
01638         };
01639     };
01640
01641     template<typename coeff, typename... coeffs>
01642     struct string_helper {
01643         static std::string func() {
01644             std::string tail = string_helper<coeffs...>::func();
01645             std::string result = "";
01646             if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
01647                 return tail;
01648             } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
01649                 if (sizeof...(coeffs) == 1) {
01650                     result += "x";
01651                 } else {
01652                     result += "x^" + std::to_string(sizeof...(coeffs));
01653                 }
01654             } else {
01655                 if (sizeof...(coeffs) == 1) {
01656                     result += coeff::to_string() + " x";
01657                 } else {
01658                     result += coeff::to_string()
+ " x^" + std::to_string(sizeof...(coeffs));
01659                 }
01660             }
01661         }
01662
01663         if (!tail.empty()) {
01664             result += " + " + tail;
01665         }
01666
01667         return result;
01668     }
01669 };
01670
01671     template<typename coeff>
01672     struct string_helper<coeff> {
01673         static std::string func() {
01674             if (!std::is_same<coeff, typename Ring::zero>::value) {
01675                 return coeff::to_string();
01676             } else {
01677                 return "";
01678             }
01679         }
01680     };
01681
01682     public:
01683     template<typename P>
01684     using simplify_t = typename simplify<P>::type;
01685
01686     template<typename v1, typename v2>
01687     using add_t = typename add<v1, v2>::type;
01688
01689     template<typename v1, typename v2>
01690     using sub_t = typename sub<v1, v2>::type;
01691
01692     template<typename v1, typename v2>
01693     using mul_t = typename mul<v1, v2>::type;
01694
01695     template<typename v1, typename v2>
01696     using eq_t = typename eq_helper<v1, v2>::type;

```

```

01711
01715     template<typename v1, typename v2>
01716     using lt_t = typename lt_helper<v1, v2>::type;
01717
01721     template<typename v1, typename v2>
01722     using gt_t = typename gt_helper<v1, v2>::type;
01723
01727     template<typename v1, typename v2>
01728     using div_t = typename div<v1, v2>::q_type;
01729
01733     template<typename v1, typename v2>
01734     using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
01735
01739     template<typename coeff, size_t deg>
01740     using monomial_t = typename monomial<coeff, deg>::type;
01741
01744     template<typename v>
01745     using derive_t = typename derive_helper<v>::type;
01746
01749     template<typename v>
01750     using pos_t = typename Ring::template pos_t<typename v::aN>;
01751
01754     template<typename v>
01755     static constexpr bool pos_v = pos_t<v>::value;
01756
01760     template<typename v1, typename v2>
01761     using gcd_t = std::conditional_t<
01762         Ring::is_euclidean_domain,
01763         typename make_unit<gcd_t<polynomial<Ring>, v1, v2>::type,
01764         void>;
01765
01769     template<auto x>
01770     using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
01771
01775     template<typename v>
01776     using inject_ring_t = val<v>;
01777 };
01778 } // namespace aerobus
01779
01780 // fraction field
01781 namespace aerobus {
01782     namespace internal {
01783         template<typename Ring, typename E = void>
01784         requires IsEuclideanDomain<Ring>
01785         struct _FractionField {};
01786
01787         template<typename Ring>
01788         requires IsEuclideanDomain<Ring>
01789         struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain>> {
01790             static constexpr bool is_field = true;
01791             static constexpr bool is_euclidean_domain = true;
01792
01793         private:
01794             template<typename val1, typename val2, typename E = void>
01795             struct to_string_helper {};
01796
01797             template<typename val1, typename val2>
01798             struct to_string_helper <val1, val2,
01799                 std::enable_if_t<
01800                     Ring::template eq_t<
01801                         val2, typename Ring::one
01802                     >::value
01803                 >
01804             > {
01805                 static std::string func() {
01806                     return val1::to_string();
01807                 }
01808             };
01809
01810             template<typename val1, typename val2>
01811             struct to_string_helper<val1, val2,
01812                 std::enable_if_t<
01813                     !Ring::template eq_t<
01814                         val2,
01815                         typename Ring::one
01816                     >::value
01817                 >
01818             > {
01819                 static std::string func() {
01820                     return "(" + val1::to_string() + " ) / ( " + val2::to_string() + " )";
01821                 }
01822             };
01823         };
01824
01825     public:
01829         template<typename val1, typename val2>
01830         struct val {
01831             using x = val1;

```

```

01834         using y = val2;
01836         using is_zero_t = typename val1::is_zero_t;
01838         static constexpr bool is_zero_v = val1::is_zero_t::value;
01839
01841         using ring_type = Ring;
01842         using enclosing_type = _FractionField<Ring>;
01843
01846         static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
01847
01851         template<typename valueType>
01852         static constexpr DEVICE INLINED valueType get() {
01853             return static_cast<valueType>(x::v) / static_cast<valueType>(y::v);
01854         }
01855
01858         static std::string to_string() {
01859             return to_string_helper<val1, val2>::func();
01860         }
01861
01866         template<typename valueRing>
01867         static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
01868             return x::eval(v) / y::eval(v);
01869         }
01870     };
01871
01873     using zero = val<typename Ring::zero, typename Ring::one>;
01875     using one = val<typename Ring::one, typename Ring::one>;
01876
01879     template<typename v>
01880     using inject_t = val<v, typename Ring::one>;
01881
01884     template<auto x>
01885     using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
Ring::one>;
01886
01889     template<typename v>
01890     using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
01891
01893     using ring_type = Ring;
01894
01895 private:
01896     template<typename v, typename E = void>
01897     struct simplify {};
01898
01899     // x = 0
01900     template<typename v>
01901     struct simplify<v, std::enable_if_t<v::x::is_zero_t::value> {
01902         using type = typename _FractionField<Ring>::zero;
01903     };
01904
01905     // x != 0
01906     template<typename v>
01907     struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value> {
01908     private:
01909         using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
01910         using newx = typename Ring::template div_t<typename v::x, _gcd>;
01911         using newy = typename Ring::template div_t<typename v::y, _gcd>;
01912
01913         using posx = std::conditional_t<
01914             !Ring::template pos_v<newy>,
01915             typename Ring::template sub_t<typename Ring::zero, newx>,
01916             newx>;
01917         using posy = std::conditional_t<
01918             !Ring::template pos_v<newy>,
01919             typename Ring::template sub_t<typename Ring::zero, newy>,
01920             newy>;
01921     public:
01922         using type = typename _FractionField<Ring>::template val<posx, posy>;
01923     };
01924
01925 public:
01926     template<typename v>
01927     using simplify_t = typename simplify<v>::type;
01928
01931 private:
01932     template<typename v1, typename v2>
01933     struct add {
01934     private:
01935         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01936         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01937         using dividend = typename Ring::template add_t<a, b>;
01938         using divider = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01939         using g = typename Ring::template gcd_t<dividend, divider>;
01940
01941     public:
01942         using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
diviser>>;
01943     };

```



```

01944
01945     template<typename v>
01946     struct pos {
01947         using type = std::conditional_t<
01948             (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
01949             (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
01950             std::true_type,
01951             std::false_type>;
01952     };
01953
01954     template<typename v1, typename v2>
01955     struct sub {
01956     private:
01957         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01958         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01959         using dividend = typename Ring::template sub_t<a, b>;
01960         using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01961         using g = typename Ring::template gcd_t<dividend, diviser>;
01962
01963     public:
01964         using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
diviser>;
01965     };
01966
01967     template<typename v1, typename v2>
01968     struct mul {
01969     private:
01970         using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
01971         using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01972
01973     public:
01974         using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
01975     };
01976
01977     template<typename v1, typename v2, typename E = void>
01978     struct div {};
01979
01980     template<typename v1, typename v2>
01981     struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
_FractionField<Ring>::zero>::value> {
01982     private:
01983         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01984         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01985
01986     public:
01987         using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
01988     };
01989
01990     template<typename v1, typename v2>
01991     struct div<v1, v2, std::enable_if_t<
std::is_same<zero, v1>::value && std::is_same<v2, zero>::value> {
01992         using type = one;
01993     };
01994
01995     template<typename v1, typename v2>
01996     struct eq {
01997     private:
01998         using type = std::conditional_t<
01999             std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
02000             std::true_type,
02001             std::false_type>;
02002
02003     };
02004
02005     template<typename v1, typename v2, typename E = void>
02006     struct gt;
02007
02008     template<typename v1, typename v2>
02009     struct gt<v1, v2, std::enable_if_t<
(eq<v1, v2>::type::value)
02010         >> {
02011         using type = std::false_type;
02012     };
02013
02014     template<typename v1, typename v2>
02015     struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02016         (!pos<v1>::type::value) && (!pos<v2>::type::value)
02017         >> {
02018         using type = typename gt<
typename sub<zero, v1>::type, typename sub<zero, v2>::type
02019             >::type;
02020     };
02021
02022     template<typename v1, typename v2>
02023     struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02024         (pos<v1>::type::value) && (!pos<v2>::type::value)
02025         >> {
02026         using type = typename gt<
typename sub<zero, v1>::type, typename sub<zero, v2>::type
02027             >::type;
02028     };

```

```

02029         » {
02030             using type = std::true_type;
02031         };
02032
02033     template<typename v1, typename v2>
02034     struct gt<v1, v2, std::enable_if_t<
02035         (!eq<v1, v2>::type::value) &&
02036         (!pos<v1>::type::value) && (pos<v2>::type::value)
02037     > {
02038         using type = std::false_type;
02039     };
02040
02041     template<typename v1, typename v2>
02042     struct gt<v1, v2, std::enable_if_t<
02043         (!eq<v1, v2>::type::value) &&
02044         (pos<v1>::type::value) && (pos<v2>::type::value)
02045     > {
02046         using type = typename Ring::template gt_t<
02047             typename Ring::template mul_t<v1::x, v2::y>,
02048             typename Ring::template mul_t<v2::y, v2::x>
02049         >;
02050     };
02051
02052     public:
02053     template<typename v1, typename v2>
02054     using add_t = typename add<v1, v2>::type;
02055
02056     template<typename v1, typename v2>
02057     using mod_t = zero;
02058
02059     template<typename v1, typename v2>
02060     using gcd_t = v1;
02061
02062     template<typename v1, typename v2>
02063     using sub_t = typename sub<v1, v2>::type;
02064
02065     template<typename v1, typename v2>
02066     using mul_t = typename mul<v1, v2>::type;
02067
02068     template<typename v1, typename v2>
02069     using div_t = typename div<v1, v2>::type;
02070
02071     template<typename v1, typename v2>
02072     using eq_t = typename eq<v1, v2>::type;
02073
02074     template<typename v1, typename v2>
02075     static constexpr bool eq_v = eq<v1, v2>::type::value;
02076
02077     template<typename v1, typename v2>
02078     using gt_t = typename gt<v1, v2>::type;
02079
02080     template<typename v1, typename v2>
02081     static constexpr bool gt_v = gt<v1, v2>::type::value;
02082
02083     template<typename v1>
02084     using pos_t = typename pos<v1>::type;
02085
02086     template<typename v>
02087     static constexpr bool pos_v = pos<t<v>::value;
02088 };
02089
02090 template<typename Ring, typename E = void>
02091 requires IsEuclideanDomain<Ring>
02092 struct FractionFieldImpl {};
02093
02094 // fraction field of a field is the field itself
02095 template<typename Field>
02096 requires IsEuclideanDomain<Field>
02097 struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field> {
02098     using type = Field;
02099     template<typename v>
02100     using inject_t = v;
02101 };
02102
02103 // fraction field of a ring is the actual fraction field
02104 template<typename Ring>
02105 requires IsEuclideanDomain<Ring>
02106 struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field> {
02107     using type = _FractionField<Ring>;
02108 };
02109 } // namespace internal
02110
02111 template<typename Ring>
02112 requires IsEuclideanDomain<Ring>
02113 using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02114
02115 template<typename Ring>

```

```

02158     struct Embed<Ring, FractionField<Ring> {
02161         template<typename v>
02162         using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02163     };
02164 } // namespace aerobus
02165
02166 // short names for common types
02167 namespace aerobus {
02171     using q32 = FractionField<i32>;
02172
02175     using fpq32 = FractionField<polynomial<q32>>;
02176
02179     using q64 = FractionField<i64>;
02180
02182     using pi64 = polynomial<i64>;
02183
02185     using pq64 = polynomial<q64>;
02186
02188     using fpq64 = FractionField<polynomial<q64>>;
02189
02194     template<typename Ring, typename v1, typename v2>
02195     using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02196
02203     template<typename v>
02204     using embed_int_poly_in_fractions_t =
02205         typename Embed<
02206             polynomial<typename v::ring_type>,
02207             polynomial<FractionField<typename v::ring_type>>::template type<v>;
02208
02212     template<int64_t p, int64_t q>
02213     using make_q64_t = typename q64::template simplify_t<
02214         typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>>;
02215
02219     template<int32_t p, int32_t q>
02220     using make_q32_t = typename q32::template simplify_t<
02221         typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>>;
02222
02227     template<typename Ring, typename v1, typename v2>
02228     using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02233     template<typename Ring, typename v1, typename v2>
02234     using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02235
02237     template<>
02238     struct Embed<q32, q64> {
02241         template<typename v>
02242         using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02243     };
02244
02248     template<typename Small, typename Large>
02249     struct Embed<polynomial<Small>, polynomial<Large> {
02250     private:
02251         template<typename v, typename i>
02252         struct at_low;
02253
02254         template<typename v, size_t i>
02255         struct at_index {
02256             using type = typename Embed<Small, Large>::template
02257             type<typename v::template coeff_at_t<i>>;
02258         };
02259
02260         template<typename v, size_t... Is>
02261         struct at_low<v, std::index_sequence<Is...> {
02262             using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
02263         };
02264     public:
02267         template<typename v>
02268         using type = typename at_low<v, typename internal::make_index_sequence_reverse<v::degree +
02269         1>::type>;
02270     };
02271
02274     template<typename Ring, auto... xs>
02275     using make_int_polynomial_t = typename polynomial<Ring>::template val<
02276         typename Ring::template inject_constant_t<xs>...>;
02277
02281     template<typename Ring, auto... xs>
02282     using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<
02283         typename FractionField<Ring>::template inject_constant_t<xs>...>;
02284 } // namespace aerobus
02285
02286 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02287 namespace aerobus {
02288     namespace internal {
02289         template<typename T, size_t x, typename E = void>
02290         struct factorial {};
02291     }

```

```

02292     template<typename T, size_t x>
02293     struct factorial<T, x, std::enable_if_t<(x > 0)>> {
02294     private:
02295         template<typename, size_t, typename>
02296         friend struct factorial;
02297     public:
02298         using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
x - 1>::type>;
02299         static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02300     };
02301
02302     template<typename T>
02303     struct factorial<T, 0> {
02304     public:
02305         using type = typename T::one;
02306         static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02307     };
02308 } // namespace internal
02309
02313 template<typename T, size_t i>
02314 using factorial_t = typename internal::factorial<T, i>::type;
02315
02319 template<typename T, size_t i>
02320 inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02321
02322 namespace internal {
02323     template<typename T, size_t k, size_t n, typename E = void>
02324     struct combination_helper {};
02325
02326     template<typename T, size_t k, size_t n>
02327     struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)>> {
02328         using type = typename FractionField<T>::template mul_t<
02329             typename combination_helper<T, k - 1, n - 1>::type,
02330             makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02331     };
02332
02333     template<typename T, size_t k, size_t n>
02334     struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)>> {
02335         using type = typename combination_helper<T, n - k, n>::type;
02336     };
02337
02338     template<typename T, size_t n>
02339     struct combination_helper<T, 0, n> {
02340         using type = typename FractionField<T>::one;
02341     };
02342
02343     template<typename T, size_t k, size_t n>
02344     struct combination {
02345         using type = typename internal::combination_helper<T, k, n>::type::x;
02346         static constexpr typename T::inner_type value =
02347             internal::combination_helper<T, k, n>::type::template get<typename
T::inner_type>();
02348     };
02349 } // namespace internal
02350
02353 template<typename T, size_t k, size_t n>
02354 using combination_t = typename internal::combination<T, k, n>::type;
02355
02360 template<typename T, size_t k, size_t n>
02361 inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02362
02363 namespace internal {
02364     template<typename T, size_t m>
02365     struct bernoulli;
02366
02367     template<typename T, typename accum, size_t k, size_t m>
02368     struct bernoulli_helper {
02369         using type = typename bernoulli_helper<
02370             T,
02371             addfractions_t<T,
02372                 accum,
02373                 mulfractions_t<T,
02374                     makefraction_t<T,
02375                         combination_t<T, k, m + 1>,
02376                         typename T::one>,
02377                         typename bernoulli<T, k>::type
02378                     >,
02379                     >,
02380                     k + 1,
02381                     m>::type;
02382     };
02383
02384     template<typename T, typename accum, size_t m>
02385     struct bernoulli_helper<T, accum, m, m> {
02386         using type = accum;

```

```

02387     };
02388
02389
02390
02391     template<typename T, size_t m>
02392     struct bernoulli {
02393         using type = typename FractionField<T>::template mul_t<
02394             typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02395             makefraction_t<T,
02396                 typename T::template val<static_cast<typename T::inner_type>(-1)>,
02397                 typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02398             >
02399         >;
02400
02401     template<typename floatType>
02402     static constexpr floatType value = type::template get<floatType>();
02403 };
02404
02405     template<typename T>
02406     struct bernoulli<T, 0> {
02407         using type = typename FractionField<T>::one;
02408
02409     template<typename floatType>
02410     static constexpr floatType value = type::template get<floatType>();
02411     };
02412 } // namespace internal
02413
02414     template<typename T, size_t n>
02415     using bernoulli_t = typename internal::bernoulli<T, n>::type;
02416
02417
02418     template<typename FloatType, typename T, size_t n>
02419     inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02420
02421 // bell numbers
02422 namespace internal {
02423     template<typename T, size_t n, typename E = void>
02424     struct bell_helper;
02425
02426     template<typename T, size_t n>
02427     struct bell_helper<T, n, std::enable_if_t<(n > 1)>> {
02428         template<typename accum, size_t i, size_t stop>
02429         struct sum_helper {
02430             private:
02431                 using left = typename T::template mul_t<
02432                     combination_t<T, i, n-1>,
02433                     typename bell_helper<T, i>::type>;
02434                 using new_accum = typename T::template add_t<accum, left>;
02435             public:
02436                 using type = typename sum_helper<new_accum, i+1, stop>::type;
02437         };
02438
02439         template<typename accum, size_t stop>
02440         struct sum_helper<accum, stop, stop> {
02441             using type = accum;
02442         };
02443
02444         using type = typename sum_helper<typename T::zero, 0, n>::type;
02445     };
02446
02447     template<typename T>
02448     struct bell_helper<T, 0> {
02449         using type = typename T::one;
02450     };
02451
02452     template<typename T>
02453     struct bell_helper<T, 1> {
02454         using type = typename T::one;
02455     };
02456 } // namespace internal
02457
02458     template<typename T, size_t n>
02459     using bell_t = typename internal::bell_helper<T, n>::type;
02460
02461     template<typename T, size_t n>
02462     static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
02463
02464     namespace internal {
02465         template<typename T, int k, typename E = void>
02466         struct alternate {};
02467
02468         template<typename T, int k>
02469         struct alternate<T, k, std::enable_if_t<k % 2 == 0>> {
02470             using type = typename T::one;
02471             static constexpr typename T::inner_type value = type::template get<typename
02472                 T::inner_type>();
02473         };
02474     }
02475
02476
02477
02478
02479

```

```

02486     template<typename T, int k>
02487     struct alternate<T, k, std::enable_if_t<k % 2 != 0>> {
02488         using type = typename T::template sub_t<typename T::zero, typename T::one>;
02489         static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02490     };
02491 } // namespace internal
02492
02493 template<typename T, int k>
02494 using alternate_t = typename internal::alternate<T, k>::type;
02495
02496 namespace internal {
02497     template<typename T, int n, int k, typename E = void>
02498     struct stirling_helper {};
02499
02500     template<typename T>
02501     struct stirling_helper<T, 0, 0> {
02502         using type = typename T::one;
02503     };
02504
02505     template<typename T, int n>
02506     struct stirling_helper<T, n, 0, std::enable_if_t<(n > 0)>> {
02507         using type = typename T::zero;
02508     };
02509
02510     template<typename T, int n>
02511     struct stirling_helper<T, 0, n, std::enable_if_t<(n > 0)>> {
02512         using type = typename T::zero;
02513     };
02514
02515     template<typename T, int n, int k>
02516     struct stirling_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)>> {
02517         using type = typename T::template sub_t<
02518             typename stirling_helper<T, n-1, k-1>::type,
02519             typename T::template mul_t<
02520                 typename T::template inject_constant_t<n-1>,
02521                 typename stirling_helper<T, n-1, k>::type
02522             >>;
02523     };
02524 } // namespace internal
02525
02526 template<typename T, int n, int k>
02527 using stirling_signed_t = typename internal::stirling_helper<T, n, k>::type;
02528
02529 template<typename T, int n, int k>
02530 using stirling_unsigned_t = abs_t<typename internal::stirling_helper<T, n, k>::type>;
02531
02532 template<typename T, int n, int k>
02533 static constexpr typename T::inner_type stirling_signed_v = stirling_signed_t<T, n, k>::v;
02534
02535 template<typename T, int n, int k>
02536 static constexpr typename T::inner_type stirling_unsigned_v = stirling_unsigned_t<T, n, k>::v;
02537
02538 template<typename T, size_t k>
02539 inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
02540
02541 namespace internal {
02542     template<typename T>
02543     struct pow_scalar {
02544         template<size_t p>
02545         static constexpr DEVICE INLINED T func(const T& x) { return p == 0 ? static_cast<T>(1) :
02546             p % 2 == 0 ? func<p/2>(x) * func<p/2>(x) :
02547             x * func<p/2>(x) * func<p/2>(x);
02548         }
02549     };
02550
02551     template<typename T, typename p, size_t n, typename E = void>
02552     requires IsEuclideanDomain<T>
02553     struct pow;
02554
02555     template<typename T, typename p, size_t n>
02556     struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)>> {
02557         using type = typename T::template mul_t<
02558             typename pow<T, p, n/2>::type,
02559             typename pow<T, p, n/2>::type
02560         >;
02561     };
02562
02563     template<typename T, typename p, size_t n>
02564     struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)>> {
02565         using type = typename T::template mul_t<
02566             p,
02567             typename T::template mul_t<
02568                 typename pow<T, p, n/2>::type,
02569                 typename pow<T, p, n/2>::type
02570             >
02571         >;
02572     };
02573 }

```

```

02592         >;
02593     };
02594
02595     template<typename T, typename p, size_t n>
02596     struct pow<T, p, n, std::enable_if_t<n == 0> { using type = typename T::one; };
02597 } // namespace internal
02598
02603     template<typename T, typename p, size_t n>
02604     using pow_t = typename internal::pow<T, p, n>::type;
02605
02610     template<typename T, typename p, size_t n>
02611     static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
02612
02613     template<typename T, size_t p>
02614     static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
internal::pow_scalar<T>::template func<p>(x); }
02615
02616     namespace internal {
02617         template<typename, template<typename, size_t> typename, class>
02618         struct make_taylor_impl;
02619
02620         template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
02621         struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...> {
02622             using type = typename polynomial<FractionField<T>::template val<typename coeff_at<T,
Is::type...>;
02623         };
02624     }
02625
02630     template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
02631     using taylor = typename internal::make_taylor_impl<
02632         T,
02633         coeff_at,
02634         internal::make_index_sequence_reverse<deg + 1>::type;
02635
02636     namespace internal {
02637         template<typename T, size_t i>
02638         struct exp_coeff {
02639             using type = makefraction_t<T, typename T::one, factorial_t<T, i>;
02640         };
02641
02642         template<typename T, size_t i, typename E = void>
02643         struct sin_coeff_helper {};
02644
02645         template<typename T, size_t i>
02646         struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02647             using type = typename FractionField<T>::zero;
02648         };
02649
02650         template<typename T, size_t i>
02651         struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02652             using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>;
02653         };
02654
02655         template<typename T, size_t i>
02656         struct sin_coeff {
02657             using type = typename sin_coeff_helper<T, i>::type;
02658         };
02659
02660         template<typename T, size_t i, typename E = void>
02661         struct sh_coeff_helper {};
02662
02663         template<typename T, size_t i>
02664         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02665             using type = typename FractionField<T>::zero;
02666         };
02667
02668         template<typename T, size_t i>
02669         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02670             using type = makefraction_t<T, typename T::one, factorial_t<T, i>;
02671         };
02672
02673         template<typename T, size_t i>
02674         struct sh_coeff {
02675             using type = typename sh_coeff_helper<T, i>::type;
02676         };
02677
02678         template<typename T, size_t i, typename E = void>
02679         struct cos_coeff_helper {};
02680
02681         template<typename T, size_t i>
02682         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02683             using type = typename FractionField<T>::zero;
02684         };
02685
02686         template<typename T, size_t i>
02687         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02688             using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>;

```

```

02689     };
02690
02691     template<typename T, size_t i>
02692     struct cos_coeff {
02693         using type = typename cos_coeff_helper<T, i>::type;
02694     };
02695
02696     template<typename T, size_t i, typename E = void>
02697     struct cosh_coeff_helper {};
02698
02699     template<typename T, size_t i>
02700     struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02701         using type = typename FractionField<T>::zero;
02702     };
02703
02704     template<typename T, size_t i>
02705     struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02706         using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02707     };
02708
02709     template<typename T, size_t i>
02710     struct cosh_coeff {
02711         using type = typename cosh_coeff_helper<T, i>::type;
02712     };
02713
02714     template<typename T, size_t i>
02715     struct geom_coeff { using type = typename FractionField<T>::one; };
02716
02717
02718     template<typename T, size_t i, typename E = void>
02719     struct atan_coeff_helper;
02720
02721     template<typename T, size_t i>
02722     struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02723         using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;
02724     };
02725
02726     template<typename T, size_t i>
02727     struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02728         using type = typename FractionField<T>::zero;
02729     };
02730
02731     template<typename T, size_t i>
02732     struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
02733
02734     template<typename T, size_t i, typename E = void>
02735     struct asin_coeff_helper;
02736
02737     template<typename T, size_t i>
02738     struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02739         using type = makefraction_t<T,
02740             factorial_t<T, i - 1>,
02741             typename T::template mul_t<
02742                 typename T::template val<i>,
02743                 T::template mul_t<
02744                     pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
02745                     pow<T, factorial_t<T, i / 2>, 2
02746                 >
02747             >
02748         >>;
02749     };
02750
02751     template<typename T, size_t i>
02752     struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02753         using type = typename FractionField<T>::zero;
02754     };
02755
02756     template<typename T, size_t i>
02757     struct asin_coeff {
02758         using type = typename asin_coeff_helper<T, i>::type;
02759     };
02760
02761     template<typename T, size_t i>
02762     struct lnpl_coeff {
02763         using type = makefraction_t<T,
02764             alternate_t<T, i + 1>,
02765             typename T::template val<i>;
02766     };
02767
02768     template<typename T>
02769     struct lnpl_coeff<T, 0> { using type = typename FractionField<T>::zero; };
02770
02771     template<typename T, size_t i, typename E = void>
02772     struct asinh_coeff_helper;
02773
02774     template<typename T, size_t i>
02775     struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {

```



```

02776         using type = makefraction_t<T,
02777             typename T::template mul_t<
02778                 alternate_t<T, i / 2>,
02779                 factorial_t<T, i - 1>
02780             >,
02781             typename T::template mul_t<
02782                 typename T::template mul_t<
02783                     typename T::template val<i>,
02784                     pow_t<T, factorial_t<T, i / 2>, 2>
02785                 >,
02786                 pow_t<T, typename T::template inject_constant_t<4>, i / 2>
02787             >
02788         >;
02789     };
02790
02791     template<typename T, size_t i>
02792     struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02793         using type = typename FractionField<T>::zero;
02794     };
02795
02796     template<typename T, size_t i>
02797     struct asinh_coeff {
02798         using type = typename asinh_coeff_helper<T, i>::type;
02799     };
02800
02801     template<typename T, size_t i, typename E = void>
02802     struct atanh_coeff_helper;
02803
02804     template<typename T, size_t i>
02805     struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02806         // 1/i
02807         using type = typename FractionField<T>::template val<
02808             typename T::one,
02809             typename T::template inject_constant_t<i>
02810         >;
02811     };
02812
02813     template<typename T, size_t i>
02814     struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02815         using type = typename FractionField<T>::zero;
02816     };
02817
02818     template<typename T, size_t i>
02819     struct atanh_coeff {
02820         using type = typename atanh_coeff_helper<T, i>::type;
02821     };
02822
02823     template<typename T, size_t i, typename E = void>
02824     struct tan_coeff_helper;
02825
02826     template<typename T, size_t i>
02827     struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0> {
02828         using type = typename FractionField<T>::zero;
02829     };
02830
02831     template<typename T, size_t i>
02832     struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0> {
02833     private:
02834         // 4^((i+1)/2)
02835         using _4p = typename FractionField<T>::template inject_t<
02836             pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2>;
02837         // 4^((i+1)/2) - 1
02838         using _4pml = typename FractionField<T>::template sub_t<_4p, typename
FractionField<T>::one>;
02839         // (-1)^((i-1)/2)
02840         using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2>;
02841         using dividend = typename FractionField<T>::template mul_t<
02842             altp,
02843             FractionField<T>::template mul_t<
02844                 _4p,
02845                 FractionField<T>::template mul_t<
02846                     _4pml,
02847                     bernoulli_t<T, (i + 1)>
02848                 >
02849             >
02850         >;
02851     public:
02852         using type = typename FractionField<T>::template div_t<dividend,
02853             typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02854     };
02855
02856     template<typename T, size_t i>
02857     struct tan_coeff {
02858         using type = typename tan_coeff_helper<T, i>::type;
02859     };
02860
02861     template<typename T, size_t i, typename E = void>
02862     struct tanh_coeff_helper;

```

```

02862
02863     template<typename T, size_t i>
02864     struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0> {
02865         using type = typename FractionField<T>::zero;
02866     };
02867
02868     template<typename T, size_t i>
02869     struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0> {
02870     private:
02871         using _4p = typename FractionField<T>::template inject_t<
02872             pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2>;
02873         using _4pml = typename FractionField<T>::template sub_t<_4p, typename
FractionField<T>::one>;
02874         using dividend =
02875             typename FractionField<T>::template mul_t<
02876                 _4p,
02877                 typename FractionField<T>::template mul_t<
02878                     _4pml,
02879                     bernoulli_t<T, (i + 1)>::type;
02880     public:
02881         using type = typename FractionField<T>::template div_t<dividend,
02882             FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02883     };
02884
02885     template<typename T, size_t i>
02886     struct tanh_coeff {
02887         using type = typename tanh_coeff_helper<T, i>::type;
02888     };
02889 } // namespace internal
02890
02891 template<typename Integers, size_t deg>
02892 using exp = taylor<Integers, internal::exp_coeff, deg>;
02893
02894 template<typename Integers, size_t deg>
02895 using expml = typename polynomial<FractionField<Integers>>::template sub_t<
02896     exp<Integers, deg>,
02897     typename polynomial<FractionField<Integers>>::one>;
02898
02899 template<typename Integers, size_t deg>
02900 using lnpl = taylor<Integers, internal::lnpl_coeff, deg>;
02901
02902 template<typename Integers, size_t deg>
02903 using atan = taylor<Integers, internal::atan_coeff, deg>;
02904
02905 template<typename Integers, size_t deg>
02906 using sin = taylor<Integers, internal::sin_coeff, deg>;
02907
02908 template<typename Integers, size_t deg>
02909 using sinh = taylor<Integers, internal::sh_coeff, deg>;
02910
02911 template<typename Integers, size_t deg>
02912 using cosh = taylor<Integers, internal::cosh_coeff, deg>;
02913
02914 template<typename Integers, size_t deg>
02915 using cos = taylor<Integers, internal::cos_coeff, deg>;
02916
02917 template<typename Integers, size_t deg>
02918 using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
02919
02920 template<typename Integers, size_t deg>
02921 using asin = taylor<Integers, internal::asin_coeff, deg>;
02922
02923 template<typename Integers, size_t deg>
02924 using asinh = taylor<Integers, internal::asinh_coeff, deg>;
02925
02926 template<typename Integers, size_t deg>
02927 using atanh = taylor<Integers, internal::atanh_coeff, deg>;
02928
02929 template<typename Integers, size_t deg>
02930 using tan = taylor<Integers, internal::tan_coeff, deg>;
02931
02932 template<typename Integers, size_t deg>
02933 using tanh = taylor<Integers, internal::tanh_coeff, deg>;
02934 } // namespace aerobus
02935
02936 // continued fractions
02937 namespace aerobus {
02938     template<int64_t... values>
02939     struct ContinuedFraction {};
02940
02941     template<int64_t a0>
02942     struct ContinuedFraction<a0> {
02943         using type = typename q64::template inject_constant_t<a0>;
02944         static constexpr double val = static_cast<double>(a0);
02945     };
02946
02947     template<int64_t a0, int64_t... rest>

```

```

03007     struct ContinuedFraction<a0, rest...> {
03009         using type = q64::template add_t<
03010             typename q64::template inject_constant_t<a0>,
03011             typename q64::template div_t<
03012                 typename q64::one,
03013                 typename ContinuedFraction<rest...>::type
03014             >>;
03015
03017         static constexpr double val = type::template get<double>();
03018     };
03019
03024     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>;
03027     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>;
03029     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>;
03031     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>;
// NOLINT
03032 } // namespace aerobus
03033
03034 // known polynomials
03035 namespace aerobus {
03036     // CChebyshev
03037     namespace internal {
03038         template<int kind, size_t deg, typename I>
03039         struct chebyshev_helper {
03040             using type = typename polynomial<I>::template sub_t<
03041                 typename polynomial<I>::template mul_t<
03042                     typename polynomial<I>::template mul_t<
03043                         typename polynomial<I>::template inject_constant_t<2>,
03044                         typename polynomial<I>::X>,
03045                         typename chebyshev_helper<kind, deg - 1, I>::type
03046                     >,
03047                     typename chebyshev_helper<kind, deg - 2, I>::type
03048                 >;
03049         };
03050
03051         template<typename I>
03052         struct chebyshev_helper<1, 0, I> {
03053             using type = typename polynomial<I>::one;
03054         };
03055
03056         template<typename I>
03057         struct chebyshev_helper<1, 1, I> {
03058             using type = typename polynomial<I>::X;
03059         };
03060
03061         template<typename I>
03062         struct chebyshev_helper<2, 0, I> {
03063             using type = typename polynomial<I>::one;
03064         };
03065
03066         template<typename I>
03067         struct chebyshev_helper<2, 1, I> {
03068             using type = typename polynomial<I>::template mul_t<
03069                 typename polynomial<I>::template inject_constant_t<2>,
03070                 typename polynomial<I>::X>;
03071         };
03072     } // namespace internal
03073
03074     // Laguerre
03075     namespace internal {
03076         template<size_t deg, typename I>
03077         struct laguerre_helper {
03078             using Q = FractionField<I>;
03079             using PQ = polynomial<Q>;
03080
03081         private:
03082             // Lk = (1 / k) * ((2 * k - 1 - x) * Lk-1 - (k - 2) Lk-2)
03083             using lnm2 = typename laguerre_helper<deg - 2, I>::type;
03084             using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03085             // -x + 2k-1
03086             using p = typename PQ::template val<
03087                 typename Q::template inject_constant_t<-1>,
03088                 typename Q::template inject_constant_t<2 * deg - 1>;
03089             // 1/n
03090             using factor = typename PQ::template inject_ring_t<
03091                 typename Q::template val<typename I::one, typename I::template
inject_constant_t<deg>>>;
03092
03093         public:
03094             using type = typename PQ::template mul_t <
03095                 factor,
03096                 typename PQ::template sub_t<
03097                     typename PQ::template mul_t<

```

```

03098         P,
03099         lnm1
03100     >,
03101     typename PQ::template mul_t<
03102         typename PQ::template inject_constant_t<deg-1>,
03103         lnm2
03104     >
03105     >
03106 >;
03107 };
03108
03109 template<typename I>
03110 struct laguerre_helper<0, I> {
03111     using type = typename polynomial<FractionField<I>::one>;
03112 };
03113
03114 template<typename I>
03115 struct laguerre_helper<1, I> {
03116     private:
03117         using PQ = polynomial<FractionField<I>;
03118     public:
03119         using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
03120 };
03121 } // namespace internal
03122
03123 // Bernstein
03124 namespace internal {
03125     template<size_t i, size_t m, typename I, typename E = void>
03126     struct bernstein_helper {};
03127
03128     template<typename I>
03129     struct bernstein_helper<0, 0, I> {
03130         using type = typename polynomial<I>::one;
03131     };
03132
03133     template<size_t i, size_t m, typename I>
03134     struct bernstein_helper<i, m, I, std::enable_if_t<
03135         (m > 0) && (i == 0)>> {
03136     private:
03137         using P = polynomial<I>;
03138     public:
03139         using type = typename P::template mul_t<
03140             typename P::template sub_t<typename P::one, typename P::X>,
03141             typename bernstein_helper<i, m-1, I>::type>;
03142     };
03143
03144     template<size_t i, size_t m, typename I>
03145     struct bernstein_helper<i, m, I, std::enable_if_t<
03146         (m > 0) && (i == m)>> {
03147     private:
03148         using P = polynomial<I>;
03149     public:
03150         using type = typename P::template mul_t<
03151             typename P::X,
03152             typename bernstein_helper<i-1, m-1, I>::type>;
03153     };
03154
03155     template<size_t i, size_t m, typename I>
03156     struct bernstein_helper<i, m, I, std::enable_if_t<
03157         (m > 0) && (i > 0) && (i < m)>> {
03158     private:
03159         using P = polynomial<I>;
03160     public:
03161         using type = typename P::template add_t<
03162             typename P::template mul_t<
03163                 typename P::template sub_t<typename P::one, typename P::X>,
03164                 typename bernstein_helper<i, m-1, I>::type>,
03165                 typename P::template mul_t<
03166                     typename P::X,
03167                     typename bernstein_helper<i-1, m-1, I>::type>;
03168     };
03169 } // namespace internal
03170
03171 namespace known_polynomials {
03172     enum hermite_kind {
03173         probabilist,
03174         physicist
03175     };
03176 }
03177
03178 // hermite
03179 namespace internal {
03180     template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03181     struct hermite_helper {};
03182
03183     template<size_t deg, typename I>
03184     struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {

```

```

03188     private:
03189         using hnm1 = typename hermite_helper<deg - 1,
known_polynomials::hermite_kind::probabilist, I>::type;
03190         using hnm2 = typename hermite_helper<deg - 2,
known_polynomials::hermite_kind::probabilist, I>::type;
03191
03192     public:
03193         using type = typename polynomial<I>::template sub_t<
03194             typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnm1>,
03195             typename polynomial<I>::template mul_t<
03196                 typename polynomial<I>::template inject_constant_t<deg - 1>,
03197                 hnm2
03198             >
03199         >;
03200     };
03201
03202     template<size_t deg, typename I>
03203     struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03204     private:
03205         using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
I>::type;
03206         using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
I>::type;
03207
03208     public:
03209         using type = typename polynomial<I>::template sub_t<
03210             // 2X Hn-1
03211             typename polynomial<I>::template mul_t<
03212                 typename pi64::val<typename I::template inject_constant_t<2>,
03213                 typename I::zero>, hnm1>,
03214
03215             typename polynomial<I>::template mul_t<
03216                 typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03217                 hnm2
03218             >
03219         >;
03220     };
03221
03222     template<typename I>
03223     struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03224     public:
03225         using type = typename polynomial<I>::one;
03226     };
03227
03228     template<typename I>
03229     struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03230     public:
03231         using type = typename polynomial<I>::X;
03232     };
03233
03234     template<typename I>
03235     struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03236     public:
03237         using type = typename pi64::one;
03238     };
03239
03240     template<typename I>
03241     struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03242     public:
03243         using type = typename polynomial<I>::template val<
03244             typename I::template inject_constant_t<2>,
03245             typename I::zero>;
03246     };
03247     } // namespace internal
03248
03249 // legendre
03250 namespace internal {
03251     template<size_t n, typename I>
03252     struct legendre_helper {
03253     private:
03254         using Q = FractionField<I>;
03255         using PQ = polynomial<Q>;
03256         // 1/n constant
03257         // (2n-1)/n X
03258         using fact_left = typename PQ::template monomial_t<
03259             makefraction_t<I,
03260             typename I::template inject_constant_t<2*n-1>,
03261             typename I::template inject_constant_t<n>
03262         >,
03263         1>;
03264         // (n-1) / n
03265         using fact_right = typename PQ::template val<
03266             makefraction_t<I,
03267             typename I::template inject_constant_t<n-1>,
03268             typename I::template inject_constant_t<n>>;
03269     public:
03270         using type = PQ::template sub_t<
03271             typename PQ::template mul_t<
03272                 fact_left,

```

```

03271         typename legendre_helper<n-1, I>::type
03272     >,
03273     typename PQ::template mul_t<
03274         fact_right,
03275         typename legendre_helper<n-2, I>::type
03276     >
03277 >;
03278 };
03279
03280 template<typename I>
03281 struct legendre_helper<0, I> {
03282     using type = typename polynomial<FractionField<I>::one;
03283 };
03284
03285 template<typename I>
03286 struct legendre_helper<1, I> {
03287     using type = typename polynomial<FractionField<I>::X;
03288 };
03289 } // namespace internal
03290
03291 // bernoulli polynomials
03292 namespace internal {
03293     template<size_t n>
03294     struct bernoulli_coeff {
03295         template<typename T, size_t i>
03296         struct inner {
03297             private:
03298                 using F = FractionField<T>;
03299             public:
03300                 using type = typename F::template mul_t<
03301                     typename F::template inject_ring_t<combination_t<T, i, n>,
03302                     bernoulli_t<T, n-i>
03303                 >;
03304         };
03305     };
03306 } // namespace internal
03307
03308 namespace known_polynomials {
03309     template <size_t deg, typename I = aerobus::i64>
03310     using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
03311
03312     template <size_t deg, typename I = aerobus::i64>
03313     using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
03314
03315     template <size_t deg, typename I = aerobus::i64>
03316     using laguerre = typename internal::laguerre_helper<deg, I>::type;
03317
03318     template <size_t deg, typename I = aerobus::i64>
03319     using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
03320 I>::type;
03321
03322     template <size_t deg, typename I = aerobus::i64>
03323     using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
03324
03325     template<size_t i, size_t m, typename I = aerobus::i64>
03326     using bernstein = typename internal::bernstein_helper<i, m, I>::type;
03327
03328     template<size_t deg, typename I = aerobus::i64>
03329     using legendre = typename internal::legendre_helper<deg, I>::type;
03330
03331     template<size_t deg, typename I = aerobus::i64>
03332     using bernoulli = taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
03333 } // namespace known_polynomials
03334 } // namespace aerobus
03335
03336 #ifdef AEROBUS_CONWAY_IMPORTS
03337 // conway polynomials
03338 namespace aerobus {
03339     template<int p, int n>
03340     struct ConwayPolynomial {};
03341
03342 #ifndef DO_NOT_DOCUMENT
03343     #define ZPZV ZPZ::template val
03344     #define POLYV aerobus::polynomial<ZPZ>::template val
03345     template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<1>; }; // NOLINT
03346     template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>; }; // NOLINT
03347     template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>; }; // NOLINT
03348     template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; }; // NOLINT
03349     template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>; }; // NOLINT
03350     template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =

```

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

[illegible]

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

[illegible]

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

[illegible]

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

[illegible]

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Chapter 10

Examples

10.1 QuotientRing

inject a 'constant' in quotient ring

inject a 'constant' in quotient ring<i32, i32::val<2>>::inject_constant_t<1>>

Template Parameters

x	a 'constant' from Ring point of view
---	--------------------------------------

10.2 type_list

A list of types <int, double, float>

A list of types <int, double, float>

Template Parameters

...Ts	types to store and manipulate at compile time
-------	---

10.3 i32::template

inject a native constant

inject a native constant

Template Parameters

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

10.4 i32::add_t

addition operator yields $v1 + v2$ $\langle i32::val\langle 2 \rangle, i32::val\langle 3 \rangle \rangle$

addition operator yields $v1 + v2$ $\langle i32::val\langle 2 \rangle, i32::val\langle 3 \rangle \rangle$

Template Parameters

$v1$	a value in i32
$v2$	a value in i32

10.5 i32::sub_t

subtraction operator yields $v1 - v2$ $\langle i32::val\langle 3 \rangle, i32::val\langle 2 \rangle \rangle$

subtraction operator yields $v1 - v2$ $\langle i32::val\langle 3 \rangle, i32::val\langle 2 \rangle \rangle$

Template Parameters

$v1$	a value in i32
$v2$	a value in i32

10.6 i32::mul_t

multiplication operator yields $v1 * v2$ $\langle i32::val\langle 3 \rangle, i32::val\langle 2 \rangle \rangle$

multiplication operator yields $v1 * v2$ $\langle i32::val\langle 3 \rangle, i32::val\langle 2 \rangle \rangle$

Template Parameters

$v1$	a value in i32
$v2$	a value in i32

10.7 i32::div_t

division operator yields $v1 / v2$ $\langle i32::val\langle 7 \rangle, i32::val\langle 2 \rangle \rangle \rightarrow i32::val\langle 3 \rangle$

division operator yields $v1 / v2$ $\langle i32::val\langle 7 \rangle, i32::val\langle 2 \rangle \rangle \rightarrow i32::val\langle 3 \rangle$

Template Parameters

$v1$	a value in i32
$v2$	a value in i32

10.8 i32::gt_t

strictly greater operator ($v1 > v2$) yields $v1 > v2$ <i32::val<7>, i32::val<2>>

strictly greater operator ($v1 > v2$) yields $v1 > v2$ <i32::val<7>, i32::val<2>>

Template Parameters

<i>v1</i>	a value in i32
<i>v2</i>	a value in i32

10.9 i32::eq_t

equality operator (type) yields $v1 == v2$ as `std::integral_constant<bool>` <i32::val<2>, i32::val<2>>

equality operator (type) yields $v1 == v2$ as `std::integral_constant<bool>` <i32::val<2>, i32::val<2>>

Template Parameters

<i>v1</i>	a value in i32
<i>v2</i>	a value in i32

10.10 i32::eq_v

equality operator (boolean value)

equality operator (boolean value)

Template Parameters

<i>v1</i>	
<i>v2</i>	<i32::val<1>, i32::val<1>>

10.11 i32::gcd_t

greatest common divisor yields $GCD(v1, v2)$ <i32::val<6>, i32::val<15>>

greatest common divisor yields $GCD(v1, v2)$ <i32::val<6>, i32::val<15>>

Template Parameters

<i>v1</i>	a value in i32
<i>v2</i>	a value in i32

10.12 i32::pos_t

positivity operator yields $v > 0$ as `std::true_type` or `std::false_type` `<i32::val<1`

positivity operator yields $v > 0$ as `std::true_type` or `std::false_type` `<i32::val<1`

Template Parameters

v	a value in i32
-----	----------------

10.13 i32::pos_v

positivity (boolean value) yields $v > 0$ as boolean value

positivity (boolean value) yields $v > 0$ as boolean value

Template Parameters

v	a value in i32 <code><i32::val<1>></code>
-----	---

10.14 i64::template

injects constant as an i64 value

injects constant as an i64 value

Template Parameters

x	<code>inject_constant_t<2></code>
-----	---

10.15 i64::add_t

addition operator

addition operator

Template Parameters

$v1$: an element of <code>aerobus::i64::val</code>
$v2$: an element of <code>aerobus::i64::val</code> <code><i64::val<1>, i64::val<2>></code>

10.16 i64::sub_t

subtraction operator

subtraction operator

Template Parameters

<i>v1</i>	: an element of aerobus::i64::val
<i>v2</i>	: an element of aerobus::i64::val <i64::val<1>, i64::val<2>>

10.17 i64::mul_t

multiplication operator

multiplication operator

Template Parameters

<i>v1</i>	: an element of aerobus::i64::val
<i>v2</i>	: an element of aerobus::i64::val <i64::val<1>, i64::val<2>>

10.18 i64::div_t

division operator integer division

division operator integer division

Template Parameters

<i>v1</i>	: an element of aerobus::i64::val
<i>v2</i>	: an element of aerobus::i64::val <i64::val<1>, i64::val<2>>

10.19 i64::mod_t

modulus operator

modulus operator

Template Parameters

<i>v1</i>	: an element of aerobus::i64::val
<i>v2</i>	: an element of aerobus::i64::val <i64::val<6>, i64::val<15>>

10.20 i64::gt_t

strictly greater operator yields $v1 > v2$ as `std::true_type` or `std::false_type`

strictly greater operator yields $v1 > v2$ as `std::true_type` or `std::false_type`

Template Parameters

<code>v1</code>	: an element of aerobus::i64::val
<code>v2</code>	: an element of aerobus::i64::val <code><i64::val<2>, i64::val<1>></code>

10.21 i64::lt_t

strict less operator yields $v1 < v2$ as `std::true_type` or `std::false_type`

strict less operator yields $v1 < v2$ as `std::true_type` or `std::false_type`

Template Parameters

<code>v1</code>	: an element of aerobus::i64::val
<code>v2</code>	: an element of aerobus::i64::val <code><i64::val<1>, i64::val<2>></code>

10.22 i64::lt_v

strictly smaller operator yields $v1 < v2$ as boolean value

strictly smaller operator yields $v1 < v2$ as boolean value

Template Parameters

<code>v1</code>	: an element of aerobus::i64::val
<code>v2</code>	: an element of aerobus::i64::val <code><i64::val<1>, i64::val<2>></code>

10.23 i64::eq_t

equality operator yields $v1 == v2$ as `std::true_type` or `std::false_type`

equality operator yields $v1 == v2$ as `std::true_type` or `std::false_type`

Template Parameters

<code>v1</code>	: an element of aerobus::i64::val
<code>v2</code>	: an element of aerobus::i64::val <code><i64::val<2>, i64::val<2>></code>

10.24 i64::eq_v

equality operator yields $v1 == v2$ as boolean value

equality operator yields $v1 == v2$ as boolean value

Template Parameters

<code>v1</code>	: an element of aerobus::i64::val
<code>v2</code>	: an element of aerobus::i64::val <i64::val<2>, i64::val<2>>

10.25 i64::gcd_t

greatest common divisor yields $GCD(v1, v2)$ as instantiation of [i64::val](#)

greatest common divisor yields $GCD(v1, v2)$ as instantiation of [i64::val](#)

Template Parameters

<code>v1</code>	: an element of aerobus::i64::val
<code>v2</code>	: an element of aerobus::i64::val <i64::val<6>, i64::val<15>>

10.26 i64::pos_t

is v positive yields $v > 0$ as [std::true_type](#) or [std::false_type](#)

is v positive yields $v > 0$ as [std::true_type](#) or [std::false_type](#)

Template Parameters

<code>v</code>	: an element of aerobus::i64::val <i64::val<1>>
----------------	---

10.27 i64::pos_v

positivity yields $v > 0$ as boolean value

positivity yields $v > 0$ as boolean value

Template Parameters

<code>v</code>	: an element of aerobus::i64::val <i64::val<1>>
----------------	---

10.28 polynomial

makes the constant (native type) polynomial `a_0`

makes the constant (native type) polynomial `a_0`

Template Parameters

<code>x</code>	<code><i32>::template inject_constant_t<2></code>
----------------	---

10.29 q32::add_t

addition operator

addition operator

Template Parameters

<code>v1</code>	a value
<code>v2</code>	a value <code><q32::val<i32::val<1>, i32::val<2>>, q32::val<i32::val<1>, i32::val<3>>></code>

10.30 FractionField

Fraction field of an euclidean domain, such as \mathbb{Q} for \mathbb{Z} .

Fraction field of an euclidean domain, such as \mathbb{Q} for \mathbb{Z}

Template Parameters

<i>Ring</i>	<code><i64></code> is q64 (rationals with 64 bits numerator and denominator)
-------------	--

10.31 PI_fraction::val

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

10.32 E_fraction::val

approximation of e -> 2.718...

approximation of e -> 2.718...

Index

abs_t
 aerobus, 21

add_t
 aerobus::i32, 58
 aerobus::i64, 62
 aerobus::polynomial< Ring >, 67
 aerobus::Quotient< Ring, X >, 74
 aerobus::zpz< p >, 98

addfractions_t
 aerobus, 21

aerobus, 17
 abs_t, 21
 addfractions_t, 21
 aligned_malloc, 33
 alternate_t, 22
 alternate_v, 34
 asin, 22
 asinh, 22
 atan, 23
 atanh, 23
 bell_t, 23
 bernoulli_t, 23
 bernoulli_v, 34
 combination_t, 24
 combination_v, 34
 cos, 24
 cosh, 24
 E_fraction, 24
 embed_int_poly_in_fractions_t, 24
 exp, 26
 expm1, 26
 factorial_t, 26
 factorial_v, 34
 field, 33
 fpq32, 26
 fpq64, 27
 FractionField, 27
 gcd_t, 27
 geometric_sum, 27
 lnp1, 27
 make_frac_polynomial_t, 28
 make_int_polynomial_t, 28
 make_q32_t, 28
 make_q64_t, 28
 makefraction_t, 29
 mulfractions_t, 29
 pi64, 29
 PI_fraction, 29
 pow_t, 30
 pq64, 30
 q32, 30
 q64, 30
 sin, 30
 sinh, 31
 SQRT2_fraction, 31
 SQRT3_fraction, 31
 stirling_signed_t, 31
 stirling_unsigned_t, 31
 tan, 32
 tanh, 32
 taylor, 32
 vadd_t, 32
 vmul_t, 33

aerobus::ContinuedFraction< a0 >, 49
 type, 49
 val, 49

aerobus::ContinuedFraction< a0, rest... >, 50
 type, 50
 val, 51

aerobus::ContinuedFraction< values >, 48

aerobus::ConwayPolynomial, 51

aerobus::Embed< i32, i64 >, 52
 type, 52

aerobus::Embed< polynomial< Small >, polynomial< Large > >, 52
 type, 53

aerobus::Embed< q32, q64 >, 53
 type, 54

aerobus::Embed< Quotient< Ring, X >, Ring >, 54
 type, 55

aerobus::Embed< Ring, FractionField< Ring > >, 55
 type, 56

aerobus::Embed< Small, Large, E >, 51

aerobus::Embed< zpz< x >, i32 >, 56
 type, 57

aerobus::i32, 57
 add_t, 58
 div_t, 58
 eq_t, 58
 eq_v, 60
 gcd_t, 58
 gt_t, 59
 inject_constant_t, 59
 inject_ring_t, 59
 inner_type, 59
 is_euclidean_domain, 60
 is_field, 60
 lt_t, 59

- mod_t, 59
- mul_t, 59
- one, 60
- pos_t, 60
- pos_v, 60
- sub_t, 60
- zero, 60
- aerobus::i32::val< x >, 82
 - enclosing_type, 83
 - eval, 84
 - get, 84
 - is_zero_t, 83
 - to_string, 84
 - v, 84
- aerobus::i64, 61
 - add_t, 62
 - div_t, 62
 - eq_t, 62
 - eq_v, 64
 - gcd_t, 62
 - gt_t, 62
 - gt_v, 64
 - inject_constant_t, 62
 - inject_ring_t, 63
 - inner_type, 63
 - is_euclidean_domain, 64
 - is_field, 64
 - lt_t, 63
 - lt_v, 64
 - mod_t, 63
 - mul_t, 63
 - one, 63
 - pos_t, 63
 - pos_v, 65
 - sub_t, 64
 - zero, 64
- aerobus::i64::val< x >, 85
 - enclosing_type, 86
 - eval, 86
 - get, 86
 - inner_type, 86
 - is_zero_t, 86
 - to_string, 87
 - v, 87
- aerobus::internal, 35
 - index_sequence_reverse, 38
 - is_instantiation_of_v, 39
 - make_index_sequence_reverse, 38
 - type_at_t, 38
- aerobus::is_prime< n >, 65
 - value, 65
- aerobus::IsEuclideanDomain, 45
- aerobus::IsField, 45
- aerobus::IsRing, 46
- aerobus::known_polynomials, 39
 - bernoulli, 40
 - bernstein, 40
 - chebyshev_T, 40
 - chebyshev_U, 41
 - hermite_kind, 43
 - hermite_phys, 41
 - hermite_prob, 41
 - laguerre, 42
 - legendre, 42
 - physicist, 43
 - probabilist, 43
- aerobus::polynomial< Ring >, 66
 - add_t, 67
 - derive_t, 68
 - div_t, 68
 - eq_t, 68
 - gcd_t, 68
 - gt_t, 69
 - inject_constant_t, 69
 - inject_ring_t, 69
 - is_euclidean_domain, 72
 - is_field, 72
 - lt_t, 69
 - mod_t, 69
 - monomial_t, 70
 - mul_t, 70
 - one, 70
 - pos_t, 70
 - pos_v, 72
 - simplify_t, 71
 - sub_t, 71
 - X, 71
 - zero, 71
- aerobus::polynomial< Ring >::val< coeffN >, 94
 - aN, 95
 - coeff_at_t, 95
 - degree, 96
 - enclosing_type, 95
 - eval, 96
 - is_zero_t, 95
 - is_zero_v, 96
 - ring_type, 95
 - strip, 95
 - to_string, 96
- aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E >, 47
- aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0 || index > 0)>, 47
- aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)>, 48
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 87
 - aN, 88
 - coeff_at_t, 88
 - degree, 90
 - enclosing_type, 89
 - eval, 89
 - is_zero_t, 89

- is_zero_v, 90
- ring_type, 89
- strip, 89
- to_string, 90
- aerobus::Quotient< Ring, X >, 73
 - add_t, 74
 - div_t, 75
 - eq_t, 75
 - eq_v, 77
 - inject_constant_t, 75
 - inject_ring_t, 75
 - is_euclidean_domain, 77
 - mod_t, 75
 - mul_t, 76
 - one, 76
 - pos_t, 76
 - pos_v, 77
 - zero, 76
- aerobus::Quotient< Ring, X >::val< V >, 91
 - raw_t, 91
 - type, 91
- aerobus::type_list< Ts >, 78
 - at, 79
 - concat, 80
 - insert, 80
 - length, 81
 - push_back, 80
 - push_front, 80
 - remove, 81
- aerobus::type_list< Ts >::pop_front, 72
 - tail, 73
 - type, 73
- aerobus::type_list< Ts >::split< index >, 78
 - head, 78
 - tail, 78
- aerobus::type_list<>, 81
 - concat, 82
 - insert, 82
 - length, 82
 - push_back, 82
 - push_front, 82
- aerobus::zpz< p >, 96
 - add_t, 98
 - div_t, 98
 - eq_t, 99
 - eq_v, 102
 - gcd_t, 99
 - gt_t, 99
 - gt_v, 102
 - inject_constant_t, 99
 - inner_type, 100
 - is_euclidean_domain, 102
 - is_field, 102
 - lt_t, 100
 - lt_v, 102
 - mod_t, 100
 - mul_t, 100
 - one, 101
 - pos_t, 101
 - pos_v, 103
 - sub_t, 101
 - zero, 101
- aerobus::zpz< p >::val< x >, 91
 - enclosing_type, 92
 - eval, 93
 - get, 93
 - is_zero_t, 92
 - is_zero_v, 93
 - to_string, 93
 - v, 93
- aligned_malloc
 - aerobus, 33
- alternate_t
 - aerobus, 22
- alternate_v
 - aerobus, 34
- aN
 - aerobus::polynomial< Ring >::val< coeffN >, 95
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 88
- asin
 - aerobus, 22
- asinh
 - aerobus, 22
- at
 - aerobus::type_list< Ts >, 79
- atan
 - aerobus, 23
- atanh
 - aerobus, 23
- bell_t
 - aerobus, 23
- bernoulli
 - aerobus::known_polynomials, 40
- bernoulli_t
 - aerobus, 23
- bernoulli_v
 - aerobus, 34
- bernstein
 - aerobus::known_polynomials, 40
- chebyshev_T
 - aerobus::known_polynomials, 40
- chebyshev_U
 - aerobus::known_polynomials, 41
- coeff_at_t
 - aerobus::polynomial< Ring >::val< coeffN >, 95
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 88
- combination_t
 - aerobus, 24
- combination_v
 - aerobus, 34
- concat
 - aerobus::type_list< Ts >, 80
 - aerobus::type_list<>, 82

- cos
 - aerobus, 24
- cosh
 - aerobus, 24
- degree
 - aerobus::polynomial< Ring >::val< coeffN >, 96
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 90
- derive_t
 - aerobus::polynomial< Ring >, 68
- div_t
 - aerobus::i32, 58
 - aerobus::i64, 62
 - aerobus::polynomial< Ring >, 68
 - aerobus::Quotient< Ring, X >, 75
 - aerobus::zpz< p >, 98
- E_fraction
 - aerobus, 24
- embed_int_poly_in_fractions_t
 - aerobus, 24
- enclosing_type
 - aerobus::i32::val< x >, 83
 - aerobus::i64::val< x >, 86
 - aerobus::polynomial< Ring >::val< coeffN >, 95
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 89
 - aerobus::zpz< p >::val< x >, 92
- eq_t
 - aerobus::i32, 58
 - aerobus::i64, 62
 - aerobus::polynomial< Ring >, 68
 - aerobus::Quotient< Ring, X >, 75
 - aerobus::zpz< p >, 99
- eq_v
 - aerobus::i32, 60
 - aerobus::i64, 64
 - aerobus::Quotient< Ring, X >, 77
 - aerobus::zpz< p >, 102
- eval
 - aerobus::i32::val< x >, 84
 - aerobus::i64::val< x >, 86
 - aerobus::polynomial< Ring >::val< coeffN >, 96
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 89
 - aerobus::zpz< p >::val< x >, 93
- exp
 - aerobus, 26
- expm1
 - aerobus, 26
- factorial_t
 - aerobus, 26
- factorial_v
 - aerobus, 34
- field
 - aerobus, 33
- fpq32
 - aerobus, 26
- fpq64
 - aerobus, 27
- FractionField
 - aerobus, 27
- gcd_t
 - aerobus, 27
 - aerobus::i32, 58
 - aerobus::i64, 62
 - aerobus::polynomial< Ring >, 68
 - aerobus::zpz< p >, 99
- geometric_sum
 - aerobus, 27
- get
 - aerobus::i32::val< x >, 84
 - aerobus::i64::val< x >, 86
 - aerobus::zpz< p >::val< x >, 93
- gt_t
 - aerobus::i32, 59
 - aerobus::i64, 62
 - aerobus::polynomial< Ring >, 69
 - aerobus::zpz< p >, 99
- gt_v
 - aerobus::i64, 64
 - aerobus::zpz< p >, 102
- head
 - aerobus::type_list< Ts >::split< index >, 78
- hermite_kind
 - aerobus::known_polynomials, 43
- hermite_phys
 - aerobus::known_polynomials, 41
- hermite_prob
 - aerobus::known_polynomials, 41
- index_sequence_reverse
 - aerobus::internal, 38
- inject_constant_t
 - aerobus::i32, 59
 - aerobus::i64, 62
 - aerobus::polynomial< Ring >, 69
 - aerobus::Quotient< Ring, X >, 75
 - aerobus::zpz< p >, 99
- inject_ring_t
 - aerobus::i32, 59
 - aerobus::i64, 63
 - aerobus::polynomial< Ring >, 69
 - aerobus::Quotient< Ring, X >, 75
- inner_type
 - aerobus::i32, 59
 - aerobus::i64, 63
 - aerobus::i64::val< x >, 86
 - aerobus::zpz< p >, 100
- insert
 - aerobus::type_list< Ts >, 80
 - aerobus::type_list<>, 82
- Introduction, 1
- is_euclidean_domain

- aerobus::i32, [60](#)
- aerobus::i64, [64](#)
- aerobus::polynomial< Ring >, [72](#)
- aerobus::Quotient< Ring, X >, [77](#)
- aerobus::zpz< p >, [102](#)
- is_field
 - aerobus::i32, [60](#)
 - aerobus::i64, [64](#)
 - aerobus::polynomial< Ring >, [72](#)
 - aerobus::zpz< p >, [102](#)
- is_instantiation_of_v
 - aerobus::internal, [39](#)
- is_zero_t
 - aerobus::i32::val< x >, [83](#)
 - aerobus::i64::val< x >, [86](#)
 - aerobus::polynomial< Ring >::val< coeffN >, [95](#)
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, [89](#)
 - aerobus::zpz< p >::val< x >, [92](#)
- is_zero_v
 - aerobus::polynomial< Ring >::val< coeffN >, [96](#)
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, [90](#)
 - aerobus::zpz< p >::val< x >, [93](#)
- laguerre
 - aerobus::known_polynomials, [42](#)
- legendre
 - aerobus::known_polynomials, [42](#)
- length
 - aerobus::type_list< Ts >, [81](#)
 - aerobus::type_list<>, [82](#)
- lnp1
 - aerobus, [27](#)
- lt_t
 - aerobus::i32, [59](#)
 - aerobus::i64, [63](#)
 - aerobus::polynomial< Ring >, [69](#)
 - aerobus::zpz< p >, [100](#)
- lt_v
 - aerobus::i64, [64](#)
 - aerobus::zpz< p >, [102](#)
- make_frac_polynomial_t
 - aerobus, [28](#)
- make_index_sequence_reverse
 - aerobus::internal, [38](#)
- make_int_polynomial_t
 - aerobus, [28](#)
- make_q32_t
 - aerobus, [28](#)
- make_q64_t
 - aerobus, [28](#)
- makefraction_t
 - aerobus, [29](#)
- mod_t
 - aerobus::i32, [59](#)
 - aerobus::i64, [63](#)
 - aerobus::polynomial< Ring >, [69](#)
- aerobus::Quotient< Ring, X >, [75](#)
- aerobus::zpz< p >, [100](#)
- monomial_t
 - aerobus::polynomial< Ring >, [70](#)
- mul_t
 - aerobus::i32, [59](#)
 - aerobus::i64, [63](#)
 - aerobus::polynomial< Ring >, [70](#)
 - aerobus::Quotient< Ring, X >, [76](#)
 - aerobus::zpz< p >, [100](#)
- mulfractions_t
 - aerobus, [29](#)
- one
 - aerobus::i32, [60](#)
 - aerobus::i64, [63](#)
 - aerobus::polynomial< Ring >, [70](#)
 - aerobus::Quotient< Ring, X >, [76](#)
 - aerobus::zpz< p >, [101](#)
- physicist
 - aerobus::known_polynomials, [43](#)
- pi64
 - aerobus, [29](#)
- PI_fraction
 - aerobus, [29](#)
- pos_t
 - aerobus::i32, [60](#)
 - aerobus::i64, [63](#)
 - aerobus::polynomial< Ring >, [70](#)
 - aerobus::Quotient< Ring, X >, [76](#)
 - aerobus::zpz< p >, [101](#)
- pos_v
 - aerobus::i32, [60](#)
 - aerobus::i64, [65](#)
 - aerobus::polynomial< Ring >, [72](#)
 - aerobus::Quotient< Ring, X >, [77](#)
 - aerobus::zpz< p >, [103](#)
- pow_t
 - aerobus, [30](#)
- pq64
 - aerobus, [30](#)
- probabilist
 - aerobus::known_polynomials, [43](#)
- push_back
 - aerobus::type_list< Ts >, [80](#)
 - aerobus::type_list<>, [82](#)
- push_front
 - aerobus::type_list< Ts >, [80](#)
 - aerobus::type_list<>, [82](#)
- q32
 - aerobus, [30](#)
- q64
 - aerobus, [30](#)
- raw_t
 - aerobus::Quotient< Ring, X >::val< V >, [91](#)
- README.md, [105](#)

remove
 aerobus::type_list< Ts >, 81
 ring_type
 aerobus::polynomial< Ring >::val< coeffN >, 95
 aerobus::polynomial< Ring >::val< coeffN, coeffs
 >, 89
 simplify_t
 aerobus::polynomial< Ring >, 71
 sin
 aerobus, 30
 sinh
 aerobus, 31
 SQRT2_fraction
 aerobus, 31
 SQRT3_fraction
 aerobus, 31
 src/aerobus.h, 105
 stirling_signed_t
 aerobus, 31
 stirling_unsigned_t
 aerobus, 31
 strip
 aerobus::polynomial< Ring >::val< coeffN >, 95
 aerobus::polynomial< Ring >::val< coeffN, coeffs
 >, 89
 sub_t
 aerobus::i32, 60
 aerobus::i64, 64
 aerobus::polynomial< Ring >, 71
 aerobus::zpz< p >, 101
 tail
 aerobus::type_list< Ts >::pop_front, 73
 aerobus::type_list< Ts >::split< index >, 78
 tan
 aerobus, 32
 tanh
 aerobus, 32
 taylor
 aerobus, 32
 to_string
 aerobus::i32::val< x >, 84
 aerobus::i64::val< x >, 87
 aerobus::polynomial< Ring >::val< coeffN >, 96
 aerobus::polynomial< Ring >::val< coeffN, coeffs
 >, 90
 aerobus::zpz< p >::val< x >, 93
 type
 aerobus::ContinuedFraction< a0 >, 49
 aerobus::ContinuedFraction< a0, rest... >, 50
 aerobus::Embed< i32, i64 >, 52
 aerobus::Embed< polynomial< Small >, polynomial< Large > >, 53
 aerobus::Embed< q32, q64 >, 54
 aerobus::Embed< Quotient< Ring, X >, Ring >, 55
 aerobus::Embed< Ring, FractionField< Ring > >, 56
 aerobus::Embed< zpz< x >, i32 >, 57
 aerobus::polynomial< Ring >::val< coeffN
 >::coeff_at< index, std::enable_if_t<(index<
 0 || index > 0)> >, 47
 aerobus::polynomial< Ring >::val< coeffN
 >::coeff_at< index, std::enable_if_t<(index==0)>
 >, 48
 aerobus::Quotient< Ring, X >::val< V >, 91
 aerobus::type_list< Ts >::pop_front, 73
 type_at_t
 aerobus::internal, 38
 v
 aerobus::i32::val< x >, 84
 aerobus::i64::val< x >, 87
 aerobus::zpz< p >::val< x >, 93
 vadd_t
 aerobus, 32
 val
 aerobus::ContinuedFraction< a0 >, 49
 aerobus::ContinuedFraction< a0, rest... >, 51
 value
 aerobus::is_prime< n >, 65
 vmul_t
 aerobus, 33
 X
 aerobus::polynomial< Ring >, 71
 zero
 aerobus::i32, 60
 aerobus::i64, 64
 aerobus::polynomial< Ring >, 71
 aerobus::Quotient< Ring, X >, 76
 aerobus::zpz< p >, 101