

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	2
1.2.1 Predefined discrete euclidean domains	2
1.2.2 Polynomials	3
1.2.3 Known polynomials	3
1.2.4 Conway polynomials	3
1.2.5 Taylor series	4
1.3 Operations	5
1.3.1 Field of fractions	5
1.3.2 Quotient	6
1.4 Misc	6
1.4.1 Continued Fractions	6
1.5 CUDA	6
2 Namespace Index	7
2.1 Namespace List	7
3 Concept Index	9
3.1 Concepts	9
4 Class Index	11
4.1 Class List	11
5 File Index	13
5.1 File List	13
6 Namespace Documentation	15
6.1 aerobus Namespace Reference	15
6.1.1 Detailed Description	19
6.1.2 Typedef Documentation	20
6.1.2.1 abs_t	20
6.1.2.2 add_t	20
6.1.2.3 addfractions_t	20
6.1.2.4 alternate_t	20
6.1.2.5 asin	21
6.1.2.6 asinh	21
6.1.2.7 atan	21
6.1.2.8 atanh	21
6.1.2.9 bell_t	23
6.1.2.10 bernoulli_t	23
6.1.2.11 combination_t	23

6.1.2.12 cos	23
6.1.2.13 cosh	25
6.1.2.14 div_t	25
6.1.2.15 E_fraction	25
6.1.2.16 embed_int_poly_in_fractions_t	25
6.1.2.17 exp	26
6.1.2.18 expm1	26
6.1.2.19 factorial_t	26
6.1.2.20 fpq32	26
6.1.2.21 fpq64	27
6.1.2.22 FractionField	27
6.1.2.23 gcd_t	27
6.1.2.24 geometric_sum	27
6.1.2.25 lnp1	28
6.1.2.26 make_frac_polynomial_t	28
6.1.2.27 make_int_polynomial_t	28
6.1.2.28 make_q32_t	28
6.1.2.29 make_q64_t	29
6.1.2.30 makefraction_t	29
6.1.2.31 mul_t	29
6.1.2.32 mulfractions_t	30
6.1.2.33 pi64	30
6.1.2.34 PI_fraction	30
6.1.2.35 pow_t	30
6.1.2.36 pq64	30
6.1.2.37 q32	31
6.1.2.38 q64	31
6.1.2.39 sin	31
6.1.2.40 sinh	31
6.1.2.41 SQRT2_fraction	31
6.1.2.42 SQRT3_fraction	32
6.1.2.43 stirling_signed_t	32
6.1.2.44 stirling_unsigned_t	32
6.1.2.45 sub_t	32
6.1.2.46 tan	33
6.1.2.47 tanh	33
6.1.2.48 taylor	33
6.1.2.49 vadd_t	33
6.1.2.50 vmul_t	34
6.1.3 Function Documentation	34
6.1.3.1 aligned_malloc()	34
6.1.3.2 field()	34

6.1.4 Variable Documentation	34
6.1.4.1 alternate_v	34
6.1.4.2 bernoulli_v	35
6.1.4.3 combination_v	35
6.1.4.4 factorial_v	35
6.2 aerobus::internal Namespace Reference	36
6.2.1 Detailed Description	39
6.2.2 Typedef Documentation	39
6.2.2.1 make_index_sequence_reverse	39
6.2.2.2 type_at_t	39
6.2.3 Function Documentation	39
6.2.3.1 index_sequence_reverse()	39
6.2.4 Variable Documentation	40
6.2.4.1 is_instantiation_of_v	40
6.3 aerobus::known_polynomials Namespace Reference	40
6.3.1 Detailed Description	40
6.3.2 Typedef Documentation	41
6.3.2.1 allone	41
6.3.2.2 bernoulli	41
6.3.2.3 bernstein	41
6.3.2.4 chebyshev_T	42
6.3.2.5 chebyshev_U	42
6.3.2.6 hermite_phys	42
6.3.2.7 hermite_prob	43
6.3.2.8 laguerre	43
6.3.2.9 legendre	43
6.3.3 Enumeration Type Documentation	44
6.3.3.1 hermite_kind	44
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	50
8.5.3.1 val	50
8.6 aerobus::ContinuedFraction< a0, rest... > Struct Template Reference	50
8.6.1 Detailed Description	50
8.6.2 Member Typedef Documentation	51
8.6.2.1 type	51
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	53
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	54
8.11.1 Detailed Description	54
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	55
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< zpz< 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::polynomial< Ring >::horner_reduction_t< P > Struct Template Reference	57
8.15.1 Detailed Description	57
8.16 aerobus::i32 Struct Reference	58
8.16.1 Detailed Description	59
8.16.2 Member Typedef Documentation	59
8.16.2.1 add_t	59
8.16.2.2 div_t	59
8.16.2.3 eq_t	59
8.16.2.4 gcd_t	60
8.16.2.5 gt_t	60
8.16.2.6 inject_constant_t	60
8.16.2.7 inject_ring_t	60
8.16.2.8 inner_type	61
8.16.2.9 lt_t	61
8.16.2.10 mod_t	61
8.16.2.11 mul_t	61
8.16.2.12 one	61
8.16.2.13 pos_t	62
8.16.2.14 sub_t	62
8.16.2.15 zero	62
8.16.3 Member Data Documentation	62
8.16.3.1 eq_v	62
8.16.3.2 is_euclidean_domain	62
8.16.3.3 is_field	63
8.16.3.4 pos_v	63
8.17 aerobus::i64 Struct Reference	63
8.17.1 Detailed Description	65
8.17.2 Member Typedef Documentation	65
8.17.2.1 add_t	65
8.17.2.2 div_t	65
8.17.2.3 eq_t	65
8.17.2.4 gcd_t	65
8.17.2.5 gt_t	66
8.17.2.6 inject_constant_t	66
8.17.2.7 inject_ring_t	66
8.17.2.8 inner_type	66

8.17.2.9 lt_t	67
8.17.2.10 mod_t	67
8.17.2.11 mul_t	67
8.17.2.12 one	67
8.17.2.13 pos_t	67
8.17.2.14 sub_t	68
8.17.2.15 zero	68
8.17.3 Member Data Documentation	68
8.17.3.1 eq_v	68
8.17.3.2 gt_v	68
8.17.3.3 is_euclidean_domain	69
8.17.3.4 is_field	69
8.17.3.5 lt_v	69
8.17.3.6 pos_v	69
8.18 aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop > Struct Template Reference	69
8.18.1 Member Typedef Documentation	70
8.18.1.1 type	70
8.19 aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< stop, stop > Struct Template Reference	70
8.19.1 Member Typedef Documentation	70
8.19.1.1 type	70
8.20 aerobus::is_prime< n > Struct Template Reference	71
8.20.1 Detailed Description	71
8.20.2 Member Data Documentation	71
8.20.2.1 value	71
8.21 aerobus::polynomial< Ring > Struct Template Reference	71
8.21.1 Detailed Description	73
8.21.2 Member Typedef Documentation	73
8.21.2.1 add_t	73
8.21.2.2 derive_t	73
8.21.2.3 div_t	74
8.21.2.4 eq_t	74
8.21.2.5 gcd_t	74
8.21.2.6 gt_t	74
8.21.2.7 inject_constant_t	75
8.21.2.8 inject_ring_t	75
8.21.2.9 lt_t	75
8.21.2.10 mod_t	75
8.21.2.11 monomial_t	76
8.21.2.12 mul_t	76
8.21.2.13 one	76
8.21.2.14 pos_t	76

8.21.2.15 simplify_t	78
8.21.2.16 sub_t	78
8.21.2.17 X	78
8.21.2.18 zero	78
8.21.3 Member Data Documentation	79
8.21.3.1 is_euclidean_domain	79
8.21.3.2 is_field	79
8.21.3.3 pos_v	79
8.22 aerobus::type_list< Ts >::pop_front Struct Reference	79
8.22.1 Detailed Description	79
8.22.2 Member Typedef Documentation	80
8.22.2.1 tail	80
8.22.2.2 type	80
8.23 aerobus::Quotient< Ring, X > Struct Template Reference	80
8.23.1 Detailed Description	81
8.23.2 Member Typedef Documentation	81
8.23.2.1 add_t	81
8.23.2.2 div_t	82
8.23.2.3 eq_t	82
8.23.2.4 inject_constant_t	82
8.23.2.5 inject_ring_t	83
8.23.2.6 mod_t	83
8.23.2.7 mul_t	83
8.23.2.8 one	83
8.23.2.9 pos_t	84
8.23.2.10 zero	84
8.23.3 Member Data Documentation	84
8.23.3.1 eq_v	84
8.23.3.2 is_euclidean_domain	84
8.23.3.3 pos_v	84
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference	85
8.24.1 Detailed Description	85
8.24.2 Member Typedef Documentation	85
8.24.2.1 head	85
8.24.2.2 tail	85
8.25 aerobus::type_list< Ts > Struct Template Reference	86
8.25.1 Detailed Description	86
8.25.2 Member Typedef Documentation	87
8.25.2.1 at	87
8.25.2.2 concat	87
8.25.2.3 insert	87
8.25.2.4 push_back	88

8.25.2.5 push_front	88
8.25.2.6 remove	88
8.25.3 Member Data Documentation	88
8.25.3.1 length	88
8.26 aerobus::type_list<> Struct Reference	89
8.26.1 Detailed Description	89
8.26.2 Member Typedef Documentation	89
8.26.2.1 concat	89
8.26.2.2 insert	89
8.26.2.3 push_back	89
8.26.2.4 push_front	89
8.26.3 Member Data Documentation	90
8.26.3.1 length	90
8.27 aerobus::i32::val< x > Struct Template Reference	90
8.27.1 Detailed Description	90
8.27.2 Member Typedef Documentation	91
8.27.2.1 enclosing_type	91
8.27.2.2 is_zero_t	91
8.27.3 Member Function Documentation	91
8.27.3.1 get()	91
8.27.3.2 to_string()	91
8.27.4 Member Data Documentation	91
8.27.4.1 v	91
8.28 aerobus::i64::val< x > Struct Template Reference	92
8.28.1 Detailed Description	92
8.28.2 Member Typedef Documentation	93
8.28.2.1 enclosing_type	93
8.28.2.2 inner_type	93
8.28.2.3 is_zero_t	93
8.28.3 Member Function Documentation	93
8.28.3.1 get()	93
8.28.3.2 to_string()	93
8.28.4 Member Data Documentation	94
8.28.4.1 v	94
8.29 aerobus::polynomial< Ring >::val< coeffN, coeffs > Struct Template Reference	94
8.29.1 Detailed Description	95
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	96
8.29.2.4 is_zero_t	96
8.29.2.5 ring_type	96

8.29.2.6 strip	96
8.29.2.7 value_at_t	96
8.29.3 Member Function Documentation	96
8.29.3.1 eval()	96
8.29.3.2 to_string()	97
8.29.4 Member Data Documentation	97
8.29.4.1 degree	97
8.29.4.2 is_zero_v	97
8.30 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference	98
8.30.1 Detailed Description	98
8.30.2 Member Typedef Documentation	98
8.30.2.1 raw_t	98
8.30.2.2 type	98
8.31 aerobus::zpz< p >::val< x > Struct Template Reference	98
8.31.1 Detailed Description	99
8.31.2 Member Typedef Documentation	99
8.31.2.1 enclosing_type	99
8.31.2.2 is_zero_t	100
8.31.3 Member Function Documentation	100
8.31.3.1 get()	100
8.31.3.2 to_string()	100
8.31.4 Member Data Documentation	100
8.31.4.1 is_zero_v	100
8.31.4.2 v	101
8.32 aerobus::polynomial< Ring >::val< coeffN > Struct Template Reference	101
8.32.1 Detailed Description	102
8.32.2 Member Typedef Documentation	103
8.32.2.1 aN	103
8.32.2.2 coeff_at_t	103
8.32.2.3 enclosing_type	103
8.32.2.4 is_zero_t	103
8.32.2.5 ring_type	103
8.32.2.6 strip	103
8.32.2.7 value_at_t	104
8.32.3 Member Function Documentation	104
8.32.3.1 eval()	104
8.32.3.2 to_string()	104
8.32.4 Member Data Documentation	104
8.32.4.1 degree	104
8.32.4.2 is_zero_v	104
8.33 aerobus::zpz< p > Struct Template Reference	105
8.33.1 Detailed Description	106

8.33.2 Member Typedef Documentation	106
8.33.2.1 add_t	106
8.33.2.2 div_t	107
8.33.2.3 eq_t	107
8.33.2.4 gcd_t	107
8.33.2.5 gt_t	108
8.33.2.6 inject_constant_t	108
8.33.2.7 inner_type	108
8.33.2.8 lt_t	108
8.33.2.9 mod_t	109
8.33.2.10 mul_t	109
8.33.2.11 one	109
8.33.2.12 pos_t	109
8.33.2.13 sub_t	110
8.33.2.14 zero	110
8.33.3 Member Data Documentation	110
8.33.3.1 eq_v	110
8.33.3.2 gt_v	110
8.33.3.3 is_euclidean_domain	111
8.33.3.4 is_field	111
8.33.3.5 lt_v	111
8.33.3.6 pos_v	111
9 File Documentation	113
9.1 README.md File Reference	113
9.2 src/aerobus.h File Reference	113
9.3 aerobus.h	113
9.4 src/examples.h File Reference	203
9.5 examples.h	203
10 Examples	205
10.1 examples/hermite.cpp	205
10.2 examples/custom_taylor.cpp	205
10.3 examples/fp16.cu	206
10.4 examples/continued_fractions.cpp	206
10.5 examples/modular_arithmetic.cpp	206
10.6 examples/make_polynomial.cpp	207
10.7 examples/polynomials_over_finite_field.cpp	207
Index	209

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 `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
```

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::zp<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 `FractionField<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
    vfmadd132pd ymm0, ymm14, ymm1
    vfmadd132pd ymm0, ymm13, ymm15
    vfmadd132pd ymm0, ymm12, ymm15
    vfmadd132pd ymm0, ymm11, ymm15
    vfmadd132pd ymm0, ymm10, ymm15
    vfmadd132pd ymm0, ymm9, ymm15
    vfmadd132pd ymm0, ymm8, ymm15
    vfmadd132pd ymm0, ymm7, ymm15
    vfmadd132pd ymm0, ymm6, ymm15
    vfmadd132pd ymm0, ymm5, ymm15
    vfmadd132pd ymm0, ymm4, ymm15
    vfmadd132pd ymm0, ymm3, ymm15
    vfmadd132pd ymm0, ymm2, ymm15
    vfmadd132pd 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 = zpz<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 `zpz<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
```

1.5 CUDA

When compiled with `nvcc` and the flag `WITH_CUDA_FP16`, `Aerobus` provides some kind of support of 16 bits integers and floats (aka `__half`).

Unfortunately, NVIDIA did not put enough `constexpr` in its `cuda_fp16.h` header, so we had to implement our own `constexpr static_cast` from `int16_t` to `__half` to make integers polynomials work with `__half`. See [this bug](#).

More, it's (at this time), not possible to make it work for `__half2` because of [another bug](#).

Please push to make these bug fixed by NVIDIA.

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	15
aerobus::internal	Internal implementations, subject to breaking changes without notice	36
aerobus::known_polynomials	Families of well known polynomials such as Hermite or Bernstein	40

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>	53
aerobus::Embed< q32, q64 >	
Embeds q32 into q64	54
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::polynomial< Ring >::horner_reduction_t< P >	
Used to evaluate polynomials over a value in Ring	57
aerobus::i32	
32 bits signed integers, seen as a algebraic ring with related operations	58
aerobus::i64	
64 bits signed integers, seen as a algebraic ring with related operations	63
aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop >	69
aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< stop, stop >	70
aerobus::is_prime< n >	
Checks if n is prime	71

aerobus::polynomial< Ring >	71
aerobus::type_list< Ts >::pop_front	
Removes types from head of the list	79
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 $\mathbb{Z}/2\mathbb{Z}$	80
aerobus::type_list< Ts >::split< index >	
Splits list at index	85
aerobus::type_list< Ts >	
Empty pure template struct to handle type list	86
aerobus::type_list<>	
Specialization for empty type list	89
aerobus::i32::val< x >	
Values in i32 , again represented as types	90
aerobus::i64::val< x >	
Values in i64	92
aerobus::polynomial< Ring >::val< coeffN, coeffs >	
Values (seen as types) in polynomial ring	94
aerobus::Quotient< Ring, X >::val< V >	
Projection values in the quotient ring	98
aerobus::zpz< p >::val< x >	
Values in zpz	98
aerobus::polynomial< Ring >::val< coeffN >	
Specialization for constants	101
aerobus::zpz< p >	
Congruence classes of integers modulo p (32 bits)	105

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

src/ aerobus.h	113
src/ examples.h	203

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 ($v_1 + v_2 + \dots + v_n$) 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
 - Fraction field of an euclidean domain, such as \mathbb{Q} for \mathbb{Z} .*
- template<typename X , typename Y >
 using [add_t](#) = typename X::enclosing_type::template [add_t](#)< X, Y >
 - generic addition*
- template<typename X , typename Y >
 using [sub_t](#) = typename X::enclosing_type::template [sub_t](#)< X, Y >

- generic subtraction*
- `template<typename X , typename Y >`
`using mul_t = typename X::enclosing_type::template mul_t< X, Y >`
- generic multiplication*
- `template<typename X , typename Y >`
`using div_t = typename X::enclosing_type::template div_t< X, Y >`
- generic division*
- `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 ln1 = taylor< Integers, internal::ln1_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 >`

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 add_t

```
template<typename X , typename Y >
using aerobus::add_t = typedef typename X::enclosing_type::template add_t<X, Y>
```

generic addition

Template Parameters

<i>X</i>	a value in a ring providing add_t operator
<i>Y</i>	a value in same ring

6.1.2.3 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 FractionField<Ring>

6.1.2.4 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
----------	---

6.1.2.5 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

6.1.2.6 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

6.1.2.7 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

6.1.2.8 atanh

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

`atanh(x)` arc hyperbolic tangent

Template Parameters

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

6.1.2.9 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

6.1.2.10 bernoulli_t

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

nth bernoulli number as type in T

Template Parameters

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

6.1.2.11 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)
----------	--

6.1.2.12 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

6.1.2.13 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

6.1.2.14 div_t

```
template<typename X , typename Y >
using aerobus::div_t = typedef typename X::enclosing_type::template div\_t<X, Y>
```

generic division

Template Parameters

<i>X</i>	a value in a euclidean domain
<i>Y</i>	a value in same Euclidean domain

6.1.2.15 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>
```

approximation of e

6.1.2.16 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>

6.1.2.17 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

6.1.2.18 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

6.1.2.19 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>	

6.1.2.20 fpq32

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

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

6.1.2.21 fpq64

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

polynomial with 64 bits rational coefficients

6.1.2.22 FractionField

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

Fraction field of an euclidean domain, such as Q for Z.

Template Parameters

<i>Ring</i>	
-------------	--

6.1.2.23 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)
----------	--------------------------------------

6.1.2.24 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

6.1.2.25 Inp1

```
template<typename Integers , size_t deg>
using aerobus::lnp1 = typedef taylor<Integers, internal::lnp1_coeff, deg>
```

$\ln(1+x)$

Template Parameters

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

6.1.2.26 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

6.1.2.27 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

6.1.2.28 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

6.1.2.29 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

6.1.2.30 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

6.1.2.31 mul_t

```
template<typename X , typename Y >
using aerobus::mul_t = typedef typename X::enclosing_type::template mul_t<X, Y>
```

generic multiplication

Template Parameters

<i>X</i>	a value in a ring providing mul_t operator
<i>Y</i>	a value in same ring

6.1.2.32 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 FransionField<Ring>

6.1.2.33 pi64

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

polynomial with 64 bits integers coefficients

6.1.2.34 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>
```

representation of π as a continued fraction

6.1.2.35 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

6.1.2.36 pq64

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

polynomial with 64 bits rationals coefficients

6.1.2.37 q32

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

32 bits rationals rationals with 32 bits numerator and denominator

6.1.2.38 q64

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

64 bits rationals rationals with 64 bits numerator and denominator

6.1.2.39 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

6.1.2.40 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

6.1.2.41 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 $\sqrt{2}$

6.1.2.46 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

6.1.2.47 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

6.1.2.48 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

<i>T</i>	Used Ring type (aerobus::i64 for example)
<i>coeff_↔ _at</i>	- implementation giving the 'value' (seen as type in FractionField<T>)
<i>deg</i>	

6.1.2.49 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>	
----------------	--

6.1.2.50 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 **AllOneHelper**
- struct **AllOneHelper**< 0, l >
- 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 **BesselHelper**
- struct **BesselHelper**< 0, l >
- struct **BesselHelper**< 1, l >
- 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 **fma_helper**
- struct **fma_helper**< double >
- struct **fma_helper**< float >
- struct **fma_helper**< int16_t >
- struct **fma_helper**< int32_t >
- struct **fma_helper**< int64_t >
- 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, I >
- struct **hermite_helper**< 1, known_polynomials::hermite_kind::physicist, I >
- struct **hermite_helper**< 1, known_polynomials::hermite_kind::probabilist, I >
- 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 **staticcast**
- 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.
- template<size_t deg, typename I = aerobus::i64>
using [allone](#) = typename internal::AllOneHelper< deg, I >::type

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 `allone`

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::allone = typedef typename internal::AllOneHelper<deg, I>↵
::type
```

6.3.2.2 `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)

6.3.2.3 `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>

6.3.2.4 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)

6.3.2.5 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>

6.3.2.6 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
------------	----------------------

6.3.2.7 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
------------	----------------------

6.3.2.8 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)

6.3.2.9 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>	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

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

Examples

[examples/continued_fractions.cpp](#).

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> representation 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> repretentation of v.

Template Parameters

<i>v</i>	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](#) repretentation of val

8.14.1 Detailed Description

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

embeds [zpz](#) values into [i32](#)

Template Parameters

<i>x</i>	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

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

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

- [src/aerobus.h](#)

8.15 aerobus::polynomial< Ring >::horner_reduction_t< P > Struct Template Reference

Used to evaluate polynomials over a value in Ring.

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

Classes

- struct [inner](#)
- struct [inner< stop, stop >](#)

8.15.1 Detailed Description

```
template<typename Ring>
template<typename P>
struct aerobus::polynomial< Ring >::horner_reduction_t< P >
```

Used to evaluate polynomials over a value in Ring.

Template Parameters

<i>P</i>	a value in polynomial<Ring>
----------	-----------------------------

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

- [src/aerobus.h](#)

8.16 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)>
inject a native constant
- template<typename v >
using [inject_ring_t](#) = v
- template<typename v1 , typename v2 >
using [add_t](#) = typename add< v1, v2 >::type
addition operator yields v1 + v2
- template<typename v1 , typename v2 >
using [sub_t](#) = typename sub< v1, v2 >::type
subtraction operator yields v1 - v2
- template<typename v1 , typename v2 >
using [mul_t](#) = typename mul< v1, v2 >::type
*multiplication operator yields v1 * v2*
- template<typename v1 , typename v2 >
using [div_t](#) = typename div< v1, v2 >::type
division operator yields v1 / v2
- template<typename v1 , typename v2 >
using [mod_t](#) = typename remainder< v1, v2 >::type
modulus operator yields v1 % v2
- template<typename v1 , typename v2 >
using [gt_t](#) = typename gt< v1, v2 >::type
strictly greater operator (v1 > v2) yields v1 > v2
- template<typename v1 , typename v2 >
using [lt_t](#) = typename lt< v1, v2 >::type
strict less operator (v1 < v2) yields v1 < v2
- template<typename v1 , typename v2 >
using [eq_t](#) = typename eq< v1, v2 >::type
equality operator (type) yields v1 == v2 as std::integral_constant<bool>
- template<typename v1 , typename v2 >
using [gcd_t](#) = [gcd_t](#)< [i32](#), v1, v2 >
greatest common divisor yields GCD(v1, v2)
- template<typename v >
using [pos_t](#) = typename pos< v >::type
positivity operator yields v > 0 as std::true_type or std::false_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
equality operator (boolean value)
- template<typename v >
static constexpr bool [pos_v](#) = [pos_t](#)<v>::value
positivity (boolean value) yields $v > 0$ as boolean value

8.16.1 Detailed Description

32 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::i32::add\_t = typename add<v1, v2>::type
```

addition operator yields $v1 + v2$

Template Parameters

v1	a value in i32
v2	a value in i32

8.16.2.2 [div_t](#)

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

division operator yields $v1 / v2$

Template Parameters

v1	a value in i32
v2	a value in i32

8.16.2.3 [eq_t](#)

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

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

Template Parameters

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

8.16.2.4 gcd_t

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

greatest common divisor yields $GCD(v1, v2)$

Template Parameters

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

8.16.2.5 gt_t

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

strictly greater operator ($v1 > v2$) yields $v1 > v2$

Template Parameters

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

8.16.2.6 inject_constant_t

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

inject a native constant

Template Parameters

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

8.16.2.7 inject_ring_t

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


8.16.2.8 inner_type

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

8.16.2.9 lt_t

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

strict less operator ($v1 < v2$) yields $v1 < v2$

Template Parameters

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

8.16.2.10 mod_t

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

modulus operator yields $v1 \% v2$

Template Parameters

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

8.16.2.11 mul_t

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

multiplication operator yields $v1 * v2$

Template Parameters

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

8.16.2.12 one

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

constant one

8.16.2.13 pos_t

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

positivity operator yields $v > 0$ as `std::true_type` or `std::false_type`

Template Parameters

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

8.16.2.14 sub_t

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

subtraction operator yields $v1 - v2$

Template Parameters

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

8.16.2.15 zero

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

equality operator (boolean value)

Template Parameters

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

8.16.3.2 is_euclidean_domain

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

integers are an euclidean domain

8.16.3.3 is_field

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

integers are not a field

8.16.3.4 pos_v

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

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

Template Parameters

<code>v</code>	a value in i32
----------------	--------------------------------

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

- src/[aerobus.h](#)

8.17 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)>
injects constant as an [i64](#) value
- 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
- 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 integer division*
 - template<typename v1 , typename v2 >
using `mod_t` = typename remainder< v1, v2 >::type
- modulus operator*
 - template<typename v1 , typename v2 >
using `gt_t` = typename gt< v1, v2 >::type
- strictly greater operator yields v1 > v2 as std::true_type or std::false_type*
 - template<typename v1 , typename v2 >
using `lt_t` = typename lt< v1, v2 >::type
- strict less operator yields v1 < v2 as std::true_type or std::false_type*
 - template<typename v1 , typename v2 >
using `eq_t` = typename eq< v1, v2 >::type
- equality operator yields v1 == v2 as std::true_type or std::false_type*
 - template<typename v1 , typename v2 >
using `gcd_t` = `gcd_t`< `i64`, v1, v2 >
- greatest common divisor yields GCD(v1, v2) as instantiation of `i64::val`*
 - template<typename v >
using `pos_t` = typename pos< v >::type
- is v positive yields v > 0 as std::true_type or std::false_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
- strictly smaller operator yields v1 < v2 as boolean value*
- template<typename v1 , typename v2 >
static constexpr bool `eq_v` = `eq_t`<v1, v2>::value
- equality operator yields v1 == v2 as boolean value*
- template<typename v >
static constexpr bool `pos_v` = `pos_t`<v>::value
- positivity yields v > 0 as boolean value*

8.17.1 Detailed Description

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

8.17.2 Member Typedef Documentation

8.17.2.1 add_t

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

addition operator

Template Parameters

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

8.17.2.2 div_t

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

division operator integer division

Template Parameters

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

8.17.2.3 eq_t

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

8.17.2.4 gcd_t

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

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

8.17.2.5 `gt_t`

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

8.17.2.6 `inject_constant_t`

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

injects constant as an [i64](#) value

Template Parameters

<code>x</code>	
----------------	--

8.17.2.7 `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 i64
----------------	--------------------------------

8.17.2.8 `inner_type`

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

type of represented values

8.17.2.9 lt_t

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

8.17.2.10 mod_t

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

modulus operator

Template Parameters

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

8.17.2.11 mul_t

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

multiplication operator

Template Parameters

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

8.17.2.12 one

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

constant one

8.17.2.13 pos_t

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

is v positive yields $v > 0$ as `std::true_type` or `std::false_type`

Template Parameters

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

8.17.2.14 `sub_t`

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

subtraction operator

Template Parameters

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

8.17.2.15 `zero`

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

constant zero

8.17.3 Member Data Documentation

8.17.3.1 `eq_v`

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

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

8.17.3.2 `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

8.17.3.3 is_euclidean_domain

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

integers are an euclidean domain

8.17.3.4 is_field

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

integers are not a field

8.17.3.5 lt_v

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

strictly smaller operator yields $v_1 < v_2$ as boolean value

Template Parameters

v_1	: an element of aerobus::i64::val
v_2	: an element of aerobus::i64::val

8.17.3.6 pos_v

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

positivity yields $v > 0$ as boolean value

Template Parameters

v	: an element of aerobus::i64::val
-----	---

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

- [src/aerobus.h](#)

8.18 aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop > Struct Template Reference

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

Public Types

- `template<typename accum , typename x >`
`using type = typename horner_reduction_t< P >::template inner< index+1, stop > ::template type< type-`
`name Ring::template add_t< typename Ring::template mul_t< x, accum >, typename P::template coeff_↵`
`at_t< P::degree - index > >, x >`

8.18.1 Member Typedef Documentation

8.18.1.1 type

```
template<typename Ring >
template<typename P >
template<size_t index, size_t stop>
template<typename accum , typename x >
using aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop >::type =
typename horner_reduction_t<P>::template inner<index + 1, stop> ::template type< typename
Ring::template add_t< typename Ring::template mul_t<x, accum>, typename P::template coeff_↵
at_t<P::degree - index> >, x>
```

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

- [src/aerobus.h](#)

8.19 aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< stop, stop > Struct Template Reference

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

Public Types

- `template<typename accum , typename x >`
`using type = accum`

8.19.1 Member Typedef Documentation

8.19.1.1 type

```
template<typename Ring >
template<typename P >
template<size_t stop>
template<typename accum , typename x >
using aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< stop, stop >::type =
accum
```

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

- [src/aerobus.h](#)

8.20 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

8.20.1 Detailed Description

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

checks if n is prime

Template Parameters

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

8.20.2 Member Data Documentation

8.20.2.1 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](#)

8.21 aerobus::polynomial< Ring > Struct Template Reference

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

Classes

- struct [horner_reduction_t](#)
Used to evaluate polynomials over a value in Ring.
- 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 > >
makes the constant (native type) polynomial a_0
- template<typename v >
using `inject_ring_t` = `val`< v >
makes the constant (ring type) polynomial a_0

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

8.21.1 Detailed Description

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

polynomial with coefficients in Ring Ring must be an integral domain

Examples

[examples/make_polynomial.cpp](#), and [examples/modular_arithmetic.cpp](#).

8.21.2 Member Typedef Documentation

8.21.2.1 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>	

8.21.2.2 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>	
----------	--

8.21.2.3 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>	

8.21.2.4 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>	

8.21.2.5 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>	

8.21.2.6 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>	

8.21.2.7 inject_constant_t

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

makes the constant (native type) polynomial a_0

Template Parameters

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

8.21.2.8 inject_ring_t

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

makes the constant (ring type) polynomial a_0

Template Parameters

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

8.21.2.9 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>	

8.21.2.10 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>	

8.21.2.11 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>	

8.21.2.12 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>	

8.21.2.13 one

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

constant one

8.21.2.14 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 ($a_n > 0$)

Template Parameters

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

8.21.2.15 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>	
----------	--

8.21.2.16 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>	

8.21.2.17 X

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

generator

8.21.2.18 zero

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

constant zero

8.21.3 Member Data Documentation

8.21.3.1 is_euclidean_domain

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

8.21.3.2 is_field

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

8.21.3.3 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](#)

8.22 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

8.22.1 Detailed Description

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

removes types from head of the list

8.22.2 Member Typedef Documentation

8.22.2.1 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

8.22.2.2 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](#)

8.23 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 > >
*inject a 'constant' in quotient ring**
 - template<typename v >
using `inject_ring_t` = val< v >
projects a value of Ring onto the quotient

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

8.23.1 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>

8.23.2 Member Typedef Documentation

8.23.2.1 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

8.23.2.2 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

8.23.2.3 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

8.23.2.4 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>
>
```

inject a 'constant' in quotient ring*

Template Parameters

<i>x</i>	a 'constant' from Ring point of view
----------	--------------------------------------

8.23.2.5 inject_ring_t

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

projects a value of Ring onto the quotient

Template Parameters

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

8.23.2.6 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

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

8.23.2.7 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

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

8.23.2.8 one

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

one

8.23.2.9 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

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

8.23.2.10 zero

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

zero value

8.23.3 Member Data Documentation

8.23.3.1 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

8.23.3.2 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

8.23.3.3 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](#)

8.24 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

8.24.1 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>	
--------------	--

8.24.2 Member Typedef Documentation

8.24.2.1 head

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

8.24.2.2 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](#)

8.25 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

8.25.1 Detailed Description

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

Empty pure template struct to handle type list.

A list of types.

Template Parameters

<i>...Ts</i>	types to store and manipulate at compile time
--------------	---

8.25.2 Member Typedef Documentation

8.25.2.1 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>	
--------------	--

8.25.2.2 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>	
----------	--

8.25.2.3 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>	

8.25.2.4 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>	
----------	--

8.25.2.5 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>	
----------	--

8.25.2.6 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>	
--------------	--

8.25.3 Member Data Documentation

8.25.3.1 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](#)

8.26 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

8.26.1 Detailed Description

specialization for empty type list

8.26.2 Member Typedef Documentation

8.26.2.1 concat

```
template<typename U >  
using aerobus::type\_list<>::concat = U
```

8.26.2.2 insert

```
template<typename T , size_t index>  
using aerobus::type\_list<>::insert = type\_list<T>
```

8.26.2.3 push_back

```
template<typename T >  
using aerobus::type\_list<>::push_back = type\_list<T>
```

8.26.2.4 push_front

```
template<typename T >  
using aerobus::type\_list<>::push_front = type\_list<T>
```

8.26.3 Member Data Documentation

8.26.3.1 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

8.27 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 [DEVICE](#) valueType [get](#) ()
cast x into valueType
- static std::string [to_string](#) ()
string representation of value

Static Public Attributes

- static constexpr int32_t [v](#) = x
actual value stored in val type

8.27.1 Detailed Description

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

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

Template Parameters

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

8.27.2 Member Typedef Documentation

8.27.2.1 enclosing_type

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

Enclosing ring type.

8.27.2.2 is_zero_t

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

is value zero

8.27.3 Member Function Documentation

8.27.3.1 get()

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

cast x into valueType

Template Parameters

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

8.27.3.2 to_string()

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

string representation of value

8.27.4 Member Data Documentation

8.27.4.1 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

8.28 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 [INLINED_DEVICE](#) valueType [get](#) ()
cast value in valueType
- static std::string [to_string](#) ()
string representation

Static Public Attributes

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

8.28.1 Detailed Description

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

values in [i64](#)

Template Parameters

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

8.28.2 Member Typedef Documentation

8.28.2.1 enclosing_type

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

enclosing ring type

8.28.2.2 inner_type

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

type of represented values

8.28.2.3 is_zero_t

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

is value zero

8.28.3 Member Function Documentation

8.28.3.1 get()

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

cast value in valueType

Template Parameters

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

8.28.3.2 to_string()

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

string representation

8.28.4 Member Data Documentation

8.28.4.1 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

8.29 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
- template<typename x >
using `value_at_t` = horner_reduction_t< val > ::template inner< 0, degree+1 > ::template type< typename Ring::zero, x >

Static Public Member Functions

- static std::string `to_string` ()
get a string representation of polynomial
- template<typename arithmeticType >
static constexpr `DEVICE INLINED` arithmeticType `eval` (const arithmeticType &x)
evaluates polynomial seen as a function operating on arithmeticType

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

8.29.1 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

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

8.29.2 Member Typedef Documentation

8.29.2.1 aN

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

heavy weight coefficient (non zero)

8.29.2.2 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>	
--------------	--

8.29.2.3 enclosing_type

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

enclosing ring type

8.29.2.4 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

8.29.2.5 ring_type

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

ring coefficients live in

8.29.2.6 strip

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

remove largest coefficient

8.29.2.7 value_at_t

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
template<typename x >
using aerobus::polynomial< Ring >::val< coeffN, coeffs >::value_at_t = horner_reduction_t<val>
::template inner<0, degree + 1> ::template type<typename Ring::zero, x>
```

8.29.3 Member Function Documentation

8.29.3.1 eval()

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

evaluates polynomial seen as a function operating on arithmeticType

Template Parameters

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

Parameters

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

Returns

$P(x)$

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

8.29.4 Member Data Documentation

8.29.4.1 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

8.29.4.2 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](#)

8.30 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 > >

8.30.1 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'
---	---------------------

8.30.2 Member Typedef Documentation

8.30.2.1 raw_t

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

8.30.2.2 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](#)

8.31 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 INLINED_DEVICE valueType get ()`
get value as valueType
- `static std::string to_string ()`
string representation

Static Public Attributes

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

8.31.1 Detailed Description

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

values in zpz

Template Parameters

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

8.31.2 Member Typedef Documentation

8.31.2.1 enclosing_type

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

enclosing ring type

8.31.2.2 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

8.31.3 Member Function Documentation

8.31.3.1 get()

```
template<int32_t p>
template<int32_t x>
template<typename valueType >
static constexpr INLINED_DEVICE 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
------------------	-----------------------------------

8.31.3.2 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

8.31.4 Member Data Documentation

8.31.4.1 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

8.31.4.2 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

8.32 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](#)
- template<typename x >
using [value_at_t](#) = [coeffN](#)

Static Public Member Functions

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

Static Public Attributes

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

8.32.1 Detailed Description

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

specialization for constants

Template Parameters

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

8.32.2 Member Typedef Documentation

8.32.2.1 aN

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

8.32.2.2 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
```

8.32.2.3 enclosing_type

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

enclosing ring type

8.32.2.4 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>
```

8.32.2.5 ring_type

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

ring coefficients live in

8.32.2.6 strip

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

8.32.2.7 value_at_t

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

8.32.3 Member Function Documentation

8.32.3.1 eval()

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

8.32.3.2 to_string()

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

8.32.4 Member Data Documentation

8.32.4.1 degree

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

degree

8.32.4.2 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](#)

8.33 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)

8.33.1 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
----------	-----------

Examples

[examples/modular_arithmetic.cpp](#), and [examples/polynomials_over_finite_field.cpp](#).

8.33.2 Member Typedef Documentation

8.33.2.1 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

8.33.2.2 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

8.33.2.3 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

8.33.2.4 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

8.33.2.5 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

8.33.2.6 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
----------	------------

8.33.2.7 inner_type

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

underlying type for values

8.33.2.8 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

8.33.2.9 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

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

8.33.2.10 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

8.33.2.11 one

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

one value

8.33.2.12 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
----	-------------------------------------

8.33.2.13 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

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

8.33.2.14 zero

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

zero value

8.33.3 Member Data Documentation

8.33.3.1 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

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

8.33.3.2 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

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

8.33.3.3 is_euclidean_domain

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

always true

8.33.3.4 is_field

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

true iff p is prime

8.33.3.5 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 (booleanvalue)

Template Parameters

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

8.33.3.6 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

v1	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 #ifdef WITH_CUDA_FP16
00016 #include <bit>
00017 #include <cuda_fp16.h>
00018 #endif
00019
00023 #ifdef _MSC_VER
00024 #define ALIGNED(x) __declspec(align(x))
00025 #define INLINED __forceinline
00026 #else
00027 #define ALIGNED(x) __attribute__((aligned(x)))
00028 #define INLINED __attribute__((always_inline)) inline
```

```

00029 #endif
00030
00031 #ifdef __CUDACC__
00032 #define DEVICE __host__ __device__
00033 #else
00034 #define DEVICE
00035 #endif
00036
00038
00040
00042
00043 // aligned allocation
00044 namespace aerobus {
00051     template<typename T>
00052     T* aligned_malloc(size_t count, size_t alignment) {
00053         #ifdef _MSC_VER
00054             return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
00055         #else
00056             return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
00057         #endif
00058     }
00059 } // namespace aerobus
00060
00061 // concepts
00062 namespace aerobus {
00064     template <typename R>
00065     concept IsRing = requires {
00066         typename R::one;
00067         typename R::zero;
00068         typename R::template add_t<typename R::one, typename R::one>;
00069         typename R::template sub_t<typename R::one, typename R::one>;
00070         typename R::template mul_t<typename R::one, typename R::one>;
00071     };
00072
00074     template <typename R>
00075     concept IsEuclideanDomain = IsRing<R> && requires {
00076         typename R::template div_t<typename R::one, typename R::one>;
00077         typename R::template mod_t<typename R::one, typename R::one>;
00078         typename R::template gcd_t<typename R::one, typename R::one>;
00079         typename R::template eq_t<typename R::one, typename R::one>;
00080         typename R::template pos_t<typename R::one>;
00081
00082         R::template pos_v<typename R::one> == true;
00083         // typename R::template gt_t<typename R::one, typename R::zero>;
00084         R::is_euclidean_domain == true;
00085     };
00086
00088     template<typename R>
00089     concept IsField = IsEuclideanDomain<R> && requires {
00090         R::is_field == true;
00091     };
00092 } // namespace aerobus
00093
00094 #ifdef WITH_CUDA_FP16
00095 // all this shit is required because of NVIDIA bug https://developer.nvidia.com/bugs/4863696
00096 namespace aerobus {
00097     namespace internal {
00098         static constexpr DEVICE uint16_t my_internal_float2half(
00099             const float f, uint32_t &sign, uint32_t &remainder) {
00100             uint32_t x;
00101             uint32_t u;
00102             uint32_t result;
00103             x = std::bit_cast<int32_t>(f);
00104             u = (x & 0x7fffffffU);
00105             sign = ((x > 0) & 0x800000U);
00106             // NaN/+Inf/-Inf
00107             if (u >= 0x7f800000U) {
00108                 remainder = 0U;
00109                 result = ((u == 0x7f800000U) ? (sign | 0x7c00U) : 0x7fffU);
00110             } else if (u > 0x477ffffU) { // Overflows
00111                 remainder = 0x80000000U;
00112                 result = (sign | 0x7bfffU);
00113             } else if (u >= 0x38800000U) { // Normal numbers
00114                 remainder = u << 19U;
00115                 u -= 0x38000000U;
00116                 result = (sign | (u >> 13U));
00117             } else if (u < 0x33000001U) { // +0/-0
00118                 remainder = u;
00119                 result = sign;
00120             } else { // Denormal numbers
00121                 const uint32_t exponent = u >> 23U;
00122                 const uint32_t shift = 0x7eU - exponent;
00123                 uint32_t mantissa = (u & 0x7ffffU);
00124                 mantissa |= 0x800000U;
00125                 remainder = mantissa << (32U - shift);
00126                 result = (sign | (mantissa >> shift));
00127                 result &= 0x0000ffffU;

```

```

00128         }
00129         return static_cast<uint16_t>(result);
00130     }
00131
00132     static constexpr DEVICE __half my_float2half_rn(const float a) {
00133         __half val;
00134         __half_raw r;
00135         uint32_t sign = 0U;
00136         uint32_t remainder = 0U;
00137         r.x = my_internal_float2half(a, sign, remainder);
00138         if ((remainder > 0x80000000U) || ((remainder == 0x80000000U) && ((r.x & 0x1U) != 0U))) {
00139             r.x++;
00140         }
00141
00142         val = std::bit_cast<__half>(r);
00143         return val;
00144     }
00145
00146     template<int16_t i>
00147     static constexpr __half convert_int16_to_half = my_float2half_rn(static_cast<float>(i));
00148
00149
00150     template<typename Out, int16_t x, typename E = void>
00151     struct int16_convert_helper;
00152
00153     template<typename Out, int16_t x>
00154     struct int16_convert_helper<Out, x,
00155         std::enable_if_t<!std::is_same_v<Out, __half> && !std::is_same_v<Out, __half2>> {
00156         static constexpr Out value() {
00157             return static_cast<Out>(x);
00158         }
00159     };
00160
00161     template<int16_t x>
00162     struct int16_convert_helper<__half, x> {
00163         static constexpr __half value() {
00164             return convert_int16_to_half<x>;
00165         }
00166     };
00167
00168     template<int16_t x>
00169     struct int16_convert_helper<__half2, x> {
00170         static constexpr __half2 value() {
00171             return __half2(convert_int16_to_half<x>, convert_int16_to_half<x>);
00172         }
00173     };
00174 } // namespace internal
00175 } // namespace aerobus
00176 #endif
00177
00178 // cast
00179 namespace aerobus {
00180     namespace internal {
00181         template<typename Out, typename In>
00182         struct staticcast {
00183             template<auto x>
00184             static constexpr INLINED_DEVICE Out func() {
00185                 return static_cast<Out>(x);
00186             }
00187         };
00188
00189         #ifdef WITH_CUDA_FP16
00190         template<>
00191         struct staticcast<__half, int16_t> {
00192             template<int16_t x>
00193             static constexpr INLINED_DEVICE __half func() {
00194                 return int16_convert_helper<__half, x>::value();
00195             }
00196         };
00197
00198         template<>
00199         struct staticcast<__half2, int16_t> {
00200             template<int16_t x>
00201             static constexpr INLINED_DEVICE __half2 func() {
00202                 return int16_convert_helper<__half2, x>::value();
00203             }
00204         };
00205         #endif
00206     } // namespace internal
00207 } // namespace aerobus
00208
00209 // fma_helper, required because nvidia fails to reconstruct fma for fp16 types
00210 namespace aerobus {
00211     namespace internal {
00212         template<typename T>
00213         struct fma_helper;
00214     }

```

```

00215     template<>
00216     struct fma_helper<double> {
00217         static constexpr INLINED_DEVICE double eval(const double x, const double y, const double
z) {
00218             return x * y + z;
00219         }
00220     };
00221
00222     template<>
00223     struct fma_helper<float> {
00224         static constexpr INLINED_DEVICE float eval(const float x, const float y, const float z) {
00225             return x * y + z;
00226         }
00227     };
00228
00229     template<>
00230     struct fma_helper<int32_t> {
00231         static constexpr INLINED_DEVICE int16_t eval(const int16_t x, const int16_t y, const
int16_t z) {
00232             return x * y + z;
00233         }
00234     };
00235
00236     template<>
00237     struct fma_helper<int16_t> {
00238         static constexpr INLINED_DEVICE int32_t eval(const int32_t x, const int32_t y, const
int32_t z) {
00239             return x * y + z;
00240         }
00241     };
00242
00243     template<>
00244     struct fma_helper<int64_t> {
00245         static constexpr INLINED_DEVICE int64_t eval(const int64_t x, const int64_t y, const
int64_t z) {
00246             return x * y + z;
00247         }
00248     };
00249
00250     #ifdef WITH_CUDA_FP16
00251     template<>
00252     struct fma_helper<__half> {
00253         static constexpr INLINED_DEVICE __half eval(const __half x, const __half y, const __half
z) {
00254             #ifdef __CUDA_ARCH__
00255                 return __hfma(x, y, z);
00256             #else
00257                 return x * y + z;
00258             #endif
00259         }
00260     };
00261     template<>
00262     struct fma_helper<__half2> {
00263         static constexpr INLINED_DEVICE __half2 eval(const __half2 x, const __half2 y, const
__half2 z) {
00264             #ifdef __CUDA_ARCH__
00265                 return __hfma2(x, y, z);
00266             #else
00267                 return x * y + z;
00268             #endif
00269         }
00270     };
00271     #endif
00272 } // namespace internal
00273 } // namespace aerobus
00274
00275 // utilities
00276 namespace aerobus {
00277     namespace internal {
00278         template<template<typename...> typename TT, typename T>
00279         struct is_instantiation_of : std::false_type { };
00280
00281         template<template<typename...> typename TT, typename... Ts>
00282         struct is_instantiation_of<TT, TT<Ts...> : std::true_type { };
00283
00284         template<template<typename...> typename TT, typename T>
00285         inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00286
00287         template<int64_t i, typename T, typename... Ts>
00288         struct type_at {
00289             static_assert(i < sizeof...(Ts) + 1, "index out of range");
00290             using type = typename type_at<i - 1, Ts...>::type;
00291         };
00292
00293         template<typename T, typename... Ts> struct type_at<0, T, Ts...> {
00294             using type = T;
00295         };

```



```

00296
00297     template <size_t i, typename... Ts>
00298     using type_at_t = typename type_at<i, Ts...>::type;
00299
00300
00301     template<size_t n, size_t i, typename E = void>
00302     struct _is_prime {};
00303
00304     template<size_t i>
00305     struct _is_prime<0, i> {
00306         static constexpr bool value = false;
00307     };
00308
00309     template<size_t i>
00310     struct _is_prime<1, i> {
00311         static constexpr bool value = false;
00312     };
00313
00314     template<size_t i>
00315     struct _is_prime<2, i> {
00316         static constexpr bool value = true;
00317     };
00318
00319     template<size_t i>
00320     struct _is_prime<3, i> {
00321         static constexpr bool value = true;
00322     };
00323
00324     template<size_t i>
00325     struct _is_prime<5, i> {
00326         static constexpr bool value = true;
00327     };
00328
00329     template<size_t i>
00330     struct _is_prime<7, i> {
00331         static constexpr bool value = true;
00332     };
00333
00334     template<size_t n, size_t i>
00335     struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)>> {
00336         static constexpr bool value = false;
00337     };
00338
00339     template<size_t n, size_t i>
00340     struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)>> {
00341         static constexpr bool value = false;
00342     };
00343
00344     template<size_t n, size_t i>
00345     struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)>> {
00346         static constexpr bool value = true;
00347     };
00348
00349     template<size_t n, size_t i>
00350     struct _is_prime<n, i, std::enable_if_t<(
00351         n % i == 0 &&
00352         n >= 9 &&
00353         n % 3 != 0 &&
00354         n % 2 != 0 &&
00355         i * i > n)>> {
00356         static constexpr bool value = true;
00357     };
00358
00359     template<size_t n, size_t i>
00360     struct _is_prime<n, i, std::enable_if_t<(
00361         n % (i+2) == 0 &&
00362         n >= 9 &&
00363         n % 3 != 0 &&
00364         n % 2 != 0 &&
00365         i * i <= n)>> {
00366         static constexpr bool value = true;
00367     };
00368
00369     template<size_t n, size_t i>
00370     struct _is_prime<n, i, std::enable_if_t<(
00371         n % (i+2) != 0 &&
00372         n % i != 0 &&
00373         n >= 9 &&
00374         n % 3 != 0 &&
00375         n % 2 != 0 &&
00376         (i * i <= n))>> {
00377         static constexpr bool value = _is_prime<n, i+6>::value;
00378     };
00379
00380 } // namespace internal
00381
00382 template<size_t n>

```

```

00385     struct is_prime {
00387         static constexpr bool value = internal::_is_prime<n, 5>::value;
00388     };
00389
00393     template<size_t n>
00394     static constexpr bool is_prime_v = is_prime<n>::value;
00395
00396     // gcd
00397     namespace internal {
00398         template <std::size_t... Is>
00399         constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
00400             -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
00401
00402         template <std::size_t N>
00403         using make_index_sequence_reverse
00404             = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00405
00411         template<typename Ring, typename E = void>
00412         struct gcd;
00413
00414         template<typename Ring>
00415         struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain> {
00416             template<typename A, typename B, typename E = void>
00417             struct gcd_helper {};
00418
00419             // B = 0, A > 0
00420             template<typename A, typename B>
00421             struct gcd_helper<A, B, std::enable_if_t<
00422                 (B::is_zero_t::value) &&
00423                 (Ring::template gt_t<A, typename Ring::zero>::value)> {
00424                 using type = A;
00425             };
00426
00427             // B = 0, A < 0
00428             template<typename A, typename B>
00429             struct gcd_helper<A, B, std::enable_if_t<
00430                 (B::is_zero_t::value) &&
00431                 !(Ring::template gt_t<A, typename Ring::zero>::value)> {
00432                 using type = typename Ring::template sub_t<typename Ring::zero, A>;
00433             };
00434
00435             // B != 0
00436             template<typename A, typename B>
00437             struct gcd_helper<A, B, std::enable_if_t<
00438                 (!B::is_zero_t::value)
00439                 > {
00440             private: // NOLINT
00441                 // A / B
00442                 using k = typename Ring::template div_t<A, B>;
00443                 // A - (A/B)*B = A % B
00444                 using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B>;
00445
00446             public:
00447                 using type = typename gcd_helper<B, m>::type;
00448             };
00449
00450             template<typename A, typename B>
00451             using type = typename gcd_helper<A, B>::type;
00452         };
00453     } // namespace internal
00454
00455     // vadd and vmul
00456     namespace internal {
00457         template<typename... vals>
00458         struct vmul {};
00459
00460         template<typename v1, typename... vals>
00461         struct vmul<v1, vals...> {
00462             using type = typename v1::enclosing_type::template mul_t<v1, typename
vmul<vals...>::type>;
00463         };
00464
00465         template<typename v1>
00466         struct vmul<v1> {
00467             using type = v1;
00468         };
00469
00470         template<typename... vals>
00471         struct vadd {};
00472
00473         template<typename v1, typename... vals>
00474         struct vadd<v1, vals...> {
00475             using type = typename v1::enclosing_type::template add_t<v1, typename
vadd<vals...>::type>;
00476         };
00477
00478         template<typename v1>

```

```

00479     struct vadd<v1> {
00480         using type = v1;
00481     };
00482 } // namespace internal
00483
00486 template<typename T, typename A, typename B>
00487 using gcd_t = typename internal::gcd<T>::template type<A, B>;
00488
00492 template<typename... vals>
00493 using vadd_t = typename internal::vadd<vals...>::type;
00494
00498 template<typename... vals>
00499 using vmul_t = typename internal::vmul<vals...>::type;
00500
00504 template<typename val>
00505 requires IsEuclideanDomain<typename val::enclosing_type>
00506 using abs_t = std::conditional_t<
00507     val::enclosing_type::template pos_v<val>,
00508     val, typename val::enclosing_type::template
00509     sub_t<typename val::enclosing_type::zero, val>>;
00510 } // namespace aerobus
00511
00511 // embedding
00512 namespace aerobus {
00513     template<typename Small, typename Large, typename E = void>
00514     struct Embed;
00515 } // namespace aerobus
00520
00521 namespace aerobus {
00522     template<typename Ring, typename X>
00523     requires IsRing<Ring>
00524     struct Quotient {
00525         template <typename V>
00526         struct val {
00527             public:
00528                 using raw_t = V;
00529                 using type = abs_t<typename Ring::template mod_t<V, X>>;
00530         };
00531
00532         using zero = val<typename Ring::zero>;
00533
00534         using one = val<typename Ring::one>;
00535
00536         template<typename v1, typename v2>
00537         using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00538
00539         template<typename v1, typename v2>
00540         using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00541
00542         template<typename v1, typename v2>
00543         using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00544
00545         template<typename v1, typename v2>
00546         using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00547
00548         template<typename v1, typename v2>
00549         using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00550
00551         template<typename v1, typename v2>
00552         static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00553
00554         template<typename v1>
00555         using pos_t = std::true_type;
00556
00557         template<typename v>
00558         static constexpr bool pos_v = pos_t<v>::value;
00559
00560         static constexpr bool is_euclidean_domain = true;
00561
00562         template<auto x>
00563         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00564
00565         template<typename v>
00566         using inject_ring_t = val<v>;
00567     };
00568
00569     template<typename Ring, typename X>
00570     struct Embed<Quotient<Ring, X>, Ring> {
00571         template<typename val>
00572         using type = typename val::raw_t;
00573     };
00574 } // namespace aerobus
00575
00576 // type_list
00577 namespace aerobus {
00578     template <typename... Ts>
00579     struct type_list;
00580 }

```



```

00728
00732     template <typename T, size_t index>
00733     using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00734
00737     template <size_t index>
00738     using remove = typename internal::remove_h<index, type_list<Ts...>>::type;
00739 };
00740
00742 template <>
00743 struct type_list<> {
00744     static constexpr size_t length = 0;
00745
00746     template <typename T>
00747     using push_front = type_list<T>;
00748
00749     template <typename T>
00750     using push_back = type_list<T>;
00751
00752     template <typename U>
00753     using concat = U;
00754
00755     // TODO(jewave): assert index == 0
00756     template <typename T, size_t index>
00757     using insert = type_list<T>;
00758 };
00759 } // namespace aerobus
00760
00761 // i16
00762 #ifdef WITH_CUDA_FP16
00763 // i16
00764 namespace aerobus {
00765     struct i16 {
00766         using inner_type = int16_t;
00767         template<int16_t x>
00770         struct val {
00771             using enclosing_type = i16;
00775             static constexpr int16_t v = x;
00776
00779             template<typename valueType>
00780             static constexpr INLINED_DEVICE valueType get() {
00781                 return internal::template int16_convert_helper<valueType, x>::value();
00782             }
00783
00785             using is_zero_t = std::bool_constant<x == 0>;
00786
00788             static std::string to_string() {
00789                 return std::to_string(x);
00790             }
00791         };
00792
00794         using zero = val<0>;
00796         using one = val<1>;
00798         static constexpr bool is_field = false;
00800         static constexpr bool is_euclidean_domain = true;
00803         template<auto x>
00804         using inject_constant_t = val<static_cast<int16_t>(x)>;
00805
00806         template<typename v>
00807         using inject_ring_t = v;
00808
00809     private:
00810         template<typename v1, typename v2>
00811         struct add {
00812             using type = val<v1::v + v2::v>;
00813         };
00814
00815         template<typename v1, typename v2>
00816         struct sub {
00817             using type = val<v1::v - v2::v>;
00818         };
00819
00820         template<typename v1, typename v2>
00821         struct mul {
00822             using type = val<v1::v * v2::v>;
00823         };
00824
00825         template<typename v1, typename v2>
00826         struct div {
00827             using type = val<v1::v / v2::v>;
00828         };
00829
00830         template<typename v1, typename v2>
00831         struct remainder {
00832             using type = val<v1::v % v2::v>;
00833         };
00834
00835         template<typename v1, typename v2>

```

```

00836     struct gt {
00837         using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00838     };
00839
00840     template<typename v1, typename v2>
00841     struct lt {
00842         using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00843     };
00844
00845     template<typename v1, typename v2>
00846     struct eq {
00847         using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00848     };
00849
00850     template<typename v1>
00851     struct pos {
00852         using type = std::bool_constant<(v1::v > 0)>;
00853     };
00854
00855 public:
00856     template<typename v1, typename v2>
00857     using add_t = typename add<v1, v2>::type;
00858
00859     template<typename v1, typename v2>
00860     using sub_t = typename sub<v1, v2>::type;
00861
00862     template<typename v1, typename v2>
00863     using mul_t = typename mul<v1, v2>::type;
00864
00865     template<typename v1, typename v2>
00866     using div_t = typename div<v1, v2>::type;
00867
00868     template<typename v1, typename v2>
00869     using mod_t = typename remainder<v1, v2>::type;
00870
00871     template<typename v1, typename v2>
00872     using gt_t = typename gt<v1, v2>::type;
00873
00874     template<typename v1, typename v2>
00875     using lt_t = typename lt<v1, v2>::type;
00876
00877     template<typename v1, typename v2>
00878     using eq_t = typename eq<v1, v2>::type;
00879
00880     template<typename v1, typename v2>
00881     static constexpr bool eq_v = eq_t<v1, v2>::value;
00882
00883     template<typename v1, typename v2>
00884     using gcd_t = gcd_t<i16, v1, v2>;
00885
00886     template<typename v>
00887     using pos_t = typename pos<v>::type;
00888
00889     template<typename v>
00890     static constexpr bool pos_v = pos_t<v>::value;
00891 };
00892 } // namespace aerobus
00893 #endif
00894
00895 // i32
00896 namespace aerobus {
00897     struct i32 {
00898         using inner_type = int32_t;
00899         template<int32_t x>
00900         struct val {
00901             using enclosing_type = i32;
00902             static constexpr int32_t v = x;
00903
00904             template<typename valueType>
00905             static constexpr DEVICE valueType get() {
00906                 return static_cast<valueType>(x);
00907             }
00908
00909             using is_zero_t = std::bool_constant<x == 0>;
00910
00911             static std::string to_string() {
00912                 return std::to_string(x);
00913             }
00914         };
00915     };
00916
00917     using zero = val<0>;
00918     using one = val<1>;
00919     static constexpr bool is_field = false;
00920     static constexpr bool is_euclidean_domain = true;
00921     template<auto x>
00922     using inject_constant_t = val<static_cast<int32_t>(x)>;
00923 }

```

```

00983     template<typename v>
00984     using inject_ring_t = v;
00985
00986 private:
00987     template<typename v1, typename v2>
00988     struct add {
00989         using type = val<v1::v + v2::v>;
00990     };
00991
00992     template<typename v1, typename v2>
00993     struct sub {
00994         using type = val<v1::v - v2::v>;
00995     };
00996
00997     template<typename v1, typename v2>
00998     struct mul {
00999         using type = val<v1::v* v2::v>;
01000     };
01001
01002     template<typename v1, typename v2>
01003     struct div {
01004         using type = val<v1::v / v2::v>;
01005     };
01006
01007     template<typename v1, typename v2>
01008     struct remainder {
01009         using type = val<v1::v % v2::v>;
01010     };
01011
01012     template<typename v1, typename v2>
01013     struct gt {
01014         using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01015     };
01016
01017     template<typename v1, typename v2>
01018     struct lt {
01019         using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01020     };
01021
01022     template<typename v1, typename v2>
01023     struct eq {
01024         using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01025     };
01026
01027     template<typename v1>
01028     struct pos {
01029         using type = std::bool_constant<(v1::v > 0)>;
01030     };
01031
01032 public:
01037     template<typename v1, typename v2>
01038     using add_t = typename add<v1, v2>::type;
01039
01044     template<typename v1, typename v2>
01045     using sub_t = typename sub<v1, v2>::type;
01046
01051     template<typename v1, typename v2>
01052     using mul_t = typename mul<v1, v2>::type;
01053
01058     template<typename v1, typename v2>
01059     using div_t = typename div<v1, v2>::type;
01060
01065     template<typename v1, typename v2>
01066     using mod_t = typename remainder<v1, v2>::type;
01067
01072     template<typename v1, typename v2>
01073     using gt_t = typename gt<v1, v2>::type;
01074
01079     template<typename v1, typename v2>
01080     using lt_t = typename lt<v1, v2>::type;
01081
01086     template<typename v1, typename v2>
01087     using eq_t = typename eq<v1, v2>::type;
01088
01092     template<typename v1, typename v2>
01093     static constexpr bool eq_v = eq_t<v1, v2>::value;
01094
01099     template<typename v1, typename v2>
01100     using gcd_t = gcd_t<i32, v1, v2>;
01101
01105     template<typename v>
01106     using pos_t = typename pos<v>::type;
01107
01111     template<typename v>
01112     static constexpr bool pos_v = pos_t<v>::value;
01113 };
01114 } // namespace aerobus

```

```

01115
01116 // i64
01117 namespace aerobus {
01118     struct i64 {
01119         using inner_type = int64_t;
01120         template<int64_t x>
01121         struct val {
01122             using inner_type = int32_t;
01123             using enclosing_type = i64;
01124             static constexpr int64_t v = x;
01125
01126             template<typename valueType>
01127             static constexpr INLINED_DEVICE valueType get() {
01128                 return static_cast<valueType>(x);
01129             }
01130
01131             using is_zero_t = std::bool_constant<x == 0>;
01132
01133             static std::string to_string() {
01134                 return std::to_string(x);
01135             }
01136         };
01137
01138         template<auto x>
01139         using inject_constant_t = val<static_cast<int64_t>(x)>;
01140
01141         template<typename v>
01142         using inject_ring_t = v;
01143
01144         using zero = val<0>;
01145         using one = val<1>;
01146         static constexpr bool is_field = false;
01147         static constexpr bool is_euclidean_domain = true;
01148
01149     private:
01150         template<typename v1, typename v2>
01151         struct add {
01152             using type = val<v1::v + v2::v>;
01153         };
01154
01155         template<typename v1, typename v2>
01156         struct sub {
01157             using type = val<v1::v - v2::v>;
01158         };
01159
01160         template<typename v1, typename v2>
01161         struct mul {
01162             using type = val<v1::v * v2::v>;
01163         };
01164
01165         template<typename v1, typename v2>
01166         struct div {
01167             using type = val<v1::v / v2::v>;
01168         };
01169
01170         template<typename v1, typename v2>
01171         struct remainder {
01172             using type = val<v1::v % v2::v>;
01173         };
01174
01175         template<typename v1, typename v2>
01176         struct gt {
01177             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01178         };
01179
01180         template<typename v1, typename v2>
01181         struct lt {
01182             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01183         };
01184
01185         template<typename v1, typename v2>
01186         struct eq {
01187             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01188         };
01189
01190         template<typename v>
01191         struct pos {
01192             using type = std::bool_constant<(v::v > 0)>;
01193         };
01194
01195     public:
01196         template<typename v1, typename v2>
01197         using add_t = typename add<v1, v2>::type;
01198
01199         template<typename v1, typename v2>
01200         using sub_t = typename sub<v1, v2>::type;
01201
01202         template<typename v1, typename v2>
01203         using mul_t = typename mul<v1, v2>::type;
01204
01205         template<typename v1, typename v2>
01206         using div_t = typename div<v1, v2>::type;
01207
01208         template<typename v1, typename v2>
01209         using remainder_t = typename remainder<v1, v2>::type;
01210
01211         template<typename v1, typename v2>
01212         using gt_t = typename gt<v1, v2>::type;
01213
01214         template<typename v1, typename v2>
01215         using lt_t = typename lt<v1, v2>::type;
01216
01217         template<typename v1, typename v2>
01218         using eq_t = typename eq<v1, v2>::type;
01219
01220         template<typename v>
01221         using pos_t = typename pos<v>::type;
01222
01223         template<typename v1, typename v2>
01224         using is_zero_t = is_zero_t;
01225
01226         template<typename v1, typename v2>
01227         using is_euclidean_domain_t = is_euclidean_domain;
01228
01229     };
01230
01231 }
01232
01233 #endif
01234
01235 // i64
01236 namespace aerobus {
01237     struct i64 {
01238         using inner_type = int64_t;
01239         template<int64_t x>
01240         struct val {
01241             using inner_type = int32_t;
01242             using enclosing_type = i64;
01243             static constexpr int64_t v = x;
01244
01245             template<typename valueType>
01246             static constexpr INLINED_DEVICE valueType get() {
01247                 return static_cast<valueType>(x);
01248             }
01249
01250             using is_zero_t = std::bool_constant<x == 0>;
01251
01252             static std::string to_string() {
01253                 return std::to_string(x);
01254             }
01255         };
01256
01257         template<auto x>
01258         using inject_constant_t = val<static_cast<int64_t>(x)>;
01259
01260         template<typename v>
01261         using inject_ring_t = v;
01262
01263         using zero = val<0>;
01264         using one = val<1>;
01265         static constexpr bool is_field = false;
01266         static constexpr bool is_euclidean_domain = true;
01267
01268     private:
01269         template<typename v1, typename v2>
01270         struct add {
01271             using type = val<v1::v + v2::v>;
01272         };
01273
01274         template<typename v1, typename v2>
01275         struct sub {
01276             using type = val<v1::v - v2::v>;
01277         };
01278
01279         template<typename v1, typename v2>
01280         struct mul {
01281             using type = val<v1::v * v2::v>;
01282         };
01283
01284         template<typename v1, typename v2>
01285         struct div {
01286             using type = val<v1::v / v2::v>;
01287         };
01288
01289         template<typename v1, typename v2>
01290         struct remainder {
01291             using type = val<v1::v % v2::v>;
01292         };
01293
01294         template<typename v1, typename v2>
01295         struct gt {
01296             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01297         };
01298
01299         template<typename v1, typename v2>
01300         struct lt {
01301             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01302         };
01303
01304         template<typename v1, typename v2>
01305         struct eq {
01306             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01307         };
01308
01309         template<typename v>
01310         struct pos {
01311             using type = std::bool_constant<(v::v > 0)>;
01312         };
01313
01314     public:
01315         template<typename v1, typename v2>
01316         using add_t = typename add<v1, v2>::type;
01317
01318         template<typename v1, typename v2>
01319         using sub_t = typename sub<v1, v2>::type;
01320
01321         template<typename v1, typename v2>
01322         using mul_t = typename mul<v1, v2>::type;
01323
01324         template<typename v1, typename v2>
01325         using div_t = typename div<v1, v2>::type;
01326
01327         template<typename v1, typename v2>
01328         using remainder_t = typename remainder<v1, v2>::type;
01329
01330         template<typename v1, typename v2>
01331         using gt_t = typename gt<v1, v2>::type;
01332
01333         template<typename v1, typename v2>
01334         using lt_t = typename lt<v1, v2>::type;
01335
01336         template<typename v1, typename v2>
01337         using eq_t = typename eq<v1, v2>::type;
01338
01339         template<typename v>
01340         using pos_t = typename pos<v>::type;
01341
01342         template<typename v1, typename v2>
01343         using is_zero_t = is_zero_t;
01344
01345         template<typename v1, typename v2>
01346         using is_euclidean_domain_t = is_euclidean_domain;
01347
01348     };
01349
01350 }
01351
01352 #endif

```



```

01232     template<typename v1, typename v2>
01233     using mul_t = typename mul<v1, v2>::type;
01234
01239     template<typename v1, typename v2>
01240     using div_t = typename div<v1, v2>::type;
01241
01245     template<typename v1, typename v2>
01246     using mod_t = typename remainder<v1, v2>::type;
01247
01252     template<typename v1, typename v2>
01253     using gt_t = typename gt<v1, v2>::type;
01254
01259     template<typename v1, typename v2>
01260     static constexpr bool gt_v = gt_t<v1, v2>::value;
01261
01266     template<typename v1, typename v2>
01267     using lt_t = typename lt<v1, v2>::type;
01268
01273     template<typename v1, typename v2>
01274     static constexpr bool lt_v = lt_t<v1, v2>::value;
01275
01280     template<typename v1, typename v2>
01281     using eq_t = typename eq<v1, v2>::type;
01282
01287     template<typename v1, typename v2>
01288     static constexpr bool eq_v = eq_t<v1, v2>::value;
01289
01294     template<typename v1, typename v2>
01295     using gcd_t = gcd_t<i64, v1, v2>;
01296
01300     template<typename v>
01301     using pos_t = typename pos<v>::type;
01302
01306     template<typename v>
01307     static constexpr bool pos_v = pos_t<v>::value;
01308 };
01309
01311 template<>
01312 struct Embed<i32, i64> {
01313     template<typename val>
01314     using type = i64::val<static_cast<int64_t>(val::v)>;
01315 };
01316 } // namespace aerobus
01317
01319 // z/pz
01320 namespace aerobus {
01321     template<int32_t p>
01322     struct zpz {
01323         using inner_type = int32_t;
01324
01325         template<int32_t x>
01326         struct val {
01327             using enclosing_type = zpz<p>;
01328             static constexpr int32_t v = x % p;
01329
01330             template<typename valueType>
01331             static constexpr INLINED_DEVICE valueType get() {
01332                 return static_cast<valueType>(x % p);
01333             }
01334
01335             using is_zero_t = std::bool_constant<v == 0>;
01336
01337             static constexpr bool is_zero_v = v == 0;
01338
01339             static std::string to_string() {
01340                 return std::to_string(x % p);
01341             }
01342         };
01343
01344         template<auto x>
01345         using inject_constant_t = val<static_cast<int32_t>(x)>;
01346
01347         using zero = val<0>;
01348
01349         using one = val<1>;
01350
01351         static constexpr bool is_field = is_prime<p>::value;
01352
01353         static constexpr bool is_euclidean_domain = true;
01354
01355     private:
01356         template<typename v1, typename v2>
01357         struct add {
01358             using type = val<(v1::v + v2::v) % p>;
01359         };
01360
01361     template<typename v1, typename v2>

```

```

01385     struct sub {
01386         using type = val<(v1::v - v2::v) % p>;
01387     };
01388
01389     template<typename v1, typename v2>
01390     struct mul {
01391         using type = val<(v1::v* v2::v) % p>;
01392     };
01393
01394     template<typename v1, typename v2>
01395     struct div {
01396         using type = val<(v1::v% p) / (v2::v % p)>;
01397     };
01398
01399     template<typename v1, typename v2>
01400     struct remainder {
01401         using type = val<(v1::v% v2::v) % p>;
01402     };
01403
01404     template<typename v1, typename v2>
01405     struct gt {
01406         using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01407     };
01408
01409     template<typename v1, typename v2>
01410     struct lt {
01411         using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01412     };
01413
01414     template<typename v1, typename v2>
01415     struct eq {
01416         using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01417     };
01418
01419     template<typename v1>
01420     struct pos {
01421         using type = std::bool_constant<(v1::v > 0)>;
01422     };
01423
01424 public:
01425     template<typename v1, typename v2>
01426     using add_t = typename add<v1, v2>::type;
01427
01428     template<typename v1, typename v2>
01429     using sub_t = typename sub<v1, v2>::type;
01430
01431     template<typename v1, typename v2>
01432     using mul_t = typename mul<v1, v2>::type;
01433
01434     template<typename v1, typename v2>
01435     using div_t = typename div<v1, v2>::type;
01436
01437     template<typename v1, typename v2>
01438     using mod_t = typename remainder<v1, v2>::type;
01439
01440     template<typename v1, typename v2>
01441     using gt_t = typename gt<v1, v2>::type;
01442
01443     template<typename v1, typename v2>
01444     static constexpr bool gt_v = gt_t<v1, v2>::value;
01445
01446     template<typename v1, typename v2>
01447     using lt_t = typename lt<v1, v2>::type;
01448
01449     template<typename v1, typename v2>
01450     static constexpr bool lt_v = lt_t<v1, v2>::value;
01451
01452     template<typename v1, typename v2>
01453     using eq_t = typename eq<v1, v2>::type;
01454
01455     template<typename v1, typename v2>
01456     static constexpr bool eq_v = eq_t<v1, v2>::value;
01457
01458     template<typename v1, typename v2>
01459     using gcd_t = gcd_t<i32, v1, v2>;
01460
01461     template<typename v1>
01462     using pos_t = typename pos<v1>::type;
01463
01464     template<typename v>
01465     static constexpr bool pos_v = pos_t<v>::value;
01466 };
01467
01468 template<int32_t x>
01469 struct Embed<zp<x>, i32> {
01470     template<typename val>
01471     using type = i32::val<val::v>;
01472 };

```

```

01516     };
01517 } // namespace aerobus
01518
01519 // polynomial
01520 namespace aerobus {
01521     // coeffN x^N + ...
01522     template<typename Ring>
01523     requires IsEuclideanDomain<Ring>
01524     struct polynomial {
01525         static constexpr bool is_field = false;
01526         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01527
01528         template<typename P>
01529         struct horner_reduction_t {
01530             template<size_t index, size_t stop>
01531             struct inner {
01532                 template<typename accum, typename x>
01533                 using type = typename horner_reduction_t<P>::template inner<index + 1, stop>
01534                     ::template type<
01535                         typename Ring::template add_t<
01536                             typename Ring::template mul_t<x, accum>,
01537                             typename P::template coeff_at_t<P::degree - index>
01538                             >, x>;
01539             };
01540
01541             template<size_t stop>
01542             struct inner<stop, stop> {
01543                 template<typename accum, typename x>
01544                 using type = accum;
01545             };
01546         };
01547
01548         template<typename coeffN, typename... coeffs>
01549         struct val {
01550             using ring_type = Ring;
01551             using enclosing_type = polynomial<Ring>;
01552             static constexpr size_t degree = sizeof...(coeffs);
01553             using aN = coeffN;
01554             using strip = val<coeffs...>;
01555             using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01556             static constexpr bool is_zero_v = is_zero_t::value;
01557
01558         private:
01559             template<size_t index, typename E = void>
01560             struct coeff_at {};
01561
01562             template<size_t index>
01563             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))> {
01564                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01565             };
01566
01567             template<size_t index>
01568             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))> {
01569                 using type = typename Ring::zero;
01570             };
01571
01572         public:
01573             template<size_t index>
01574             using coeff_at_t = typename coeff_at<index>::type;
01575
01576             static std::string to_string() {
01577                 return string_helper<coeffN, coeffs...>::func();
01578             }
01579
01580             template<typename arithmeticType>
01581             static constexpr DEVICE INLINEED arithmeticType eval(const arithmeticType& x) {
01582                 #ifdef WITH_CUDA_FP16
01583                 arithmeticType start;
01584                 if constexpr (std::is_same_v<arithmeticType, __half2>) {
01585                     start = __half2(0, 0);
01586                 } else {
01587                     start = static_cast<arithmeticType>(0);
01588                 }
01589                 #else
01590                 arithmeticType start = static_cast<arithmeticType>(0);
01591                 #endif
01592                 return horner_evaluation<arithmeticType, val>
01593                     ::template inner<0, degree + 1>
01594                     ::func(start, x);
01595             }
01596
01597             template<typename x>
01598             using value_at_t = horner_reduction_t<val>
01599                 ::template inner<0, degree + 1>
01600                 ::template type<typename Ring::zero, x>;
01601         };
01602     };
01603 }

```

```

01629     template<typename coeffN>
01630     struct val<coeffN> {
01631         using ring_type = Ring;
01632         using enclosing_type = polynomial<Ring>;
01633         static constexpr size_t degree = 0;
01634         using aN = coeffN;
01635         using strip = val<coeffN>;
01636         using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01637
01638         static constexpr bool is_zero_v = is_zero_t::value;
01639
01640         template<size_t index, typename E = void>
01641         struct coeff_at {};
01642
01643         template<size_t index>
01644         struct coeff_at<index, std::enable_if_t<(index == 0)>> {
01645             using type = aN;
01646         };
01647
01648         template<size_t index>
01649         struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)>> {
01650             using type = typename Ring::zero;
01651         };
01652
01653         template<size_t index>
01654         using coeff_at_t = typename coeff_at<index>::type;
01655
01656         static std::string to_string() {
01657             return string_helper<coeffN>::func();
01658         }
01659
01660         template<typename arithmeticType>
01661         static constexpr DEVICE INLINE arithmeticType eval(const arithmeticType& x) {
01662             return coeffN::template get<arithmeticType>();
01663         }
01664
01665         template<typename x>
01666         using value_at_t = coeffN;
01667     };
01668
01669     using zero = val<typename Ring::zero>;
01670     using one = val<typename Ring::one>;
01671     using X = val<typename Ring::one, typename Ring::zero>;
01672
01673 private:
01674     template<typename P, typename E = void>
01675     struct simplify;
01676
01677     template <typename P1, typename P2, typename I>
01678     struct add_low;
01679
01680     template<typename P1, typename P2>
01681     struct add {
01682         using type = typename simplify<typename add_low<
01683             P1,
01684             P2,
01685             internal::make_index_sequence_reverse<
01686                 std::max(P1::degree, P2::degree) + 1
01687             >::type>::type;
01688     };
01689
01690     template <typename P1, typename P2, typename I>
01691     struct sub_low;
01692
01693     template <typename P1, typename P2, typename I>
01694     struct mul_low;
01695
01696     template<typename v1, typename v2>
01697     struct mul {
01698         using type = typename mul_low<
01699             v1,
01700             v2,
01701             internal::make_index_sequence_reverse<
01702                 v1::degree + v2::degree + 1
01703             >::type;
01704     };
01705
01706     template<typename coeff, size_t deg>
01707     struct monomial;
01708
01709     template<typename v, typename E = void>
01710     struct derive_helper {};
01711
01712     template<typename v>
01713     struct derive_helper<v, std::enable_if_t<v::degree == 0>> {
01714         using type = zero;
01715     };
01716

```

```

01722
01723     template<typename v>
01724     struct derive_helper<v, std::enable_if_t<v::degree != 0> {
01725         using type = typename add<
01726             typename derive_helper<typename simplify<typename v::strip>::type>::type,
01727             typename monomial<
01728                 typename Ring::template mul_t<
01729                     typename v::aN,
01730                     typename Ring::template inject_constant_t<(v::degree)>
01731                 >,
01732                 v::degree - 1
01733             >::type
01734         >::type;
01735     };
01736
01737     template<typename v1, typename v2, typename E = void>
01738     struct eq_helper {};
01739
01740     template<typename v1, typename v2>
01741     struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree> {
01742         using type = std::false_type;
01743     };
01744
01745     template<typename v1, typename v2>
01746     struct eq_helper<v1, v2, std::enable_if_t<
01747         v1::degree == v2::degree &&
01748         (v1::degree != 0 || v2::degree != 0) &&
01749         std::is_same<
01750             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01751             std::false_type
01752         >::value
01753     >
01754     > {
01755     > {
01756         using type = std::false_type;
01757     };
01758
01759     template<typename v1, typename v2>
01760     struct eq_helper<v1, v2, std::enable_if_t<
01761         v1::degree == v2::degree &&
01762         (v1::degree != 0 || v2::degree != 0) &&
01763         std::is_same<
01764             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01765             std::true_type
01766         >::value
01767     > {
01768         using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01769     };
01770
01771     template<typename v1, typename v2>
01772     struct eq_helper<v1, v2, std::enable_if_t<
01773         v1::degree == v2::degree &&
01774         (v1::degree == 0)
01775     > {
01776         using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01777     };
01778
01779     template<typename v1, typename v2, typename E = void>
01780     struct lt_helper {};
01781
01782     template<typename v1, typename v2>
01783     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)> {
01784         using type = std::true_type;
01785     };
01786
01787     template<typename v1, typename v2>
01788     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)> {
01789         using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01790     };
01791
01792     template<typename v1, typename v2>
01793     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)> {
01794         using type = std::false_type;
01795     };
01796
01797     template<typename v1, typename v2, typename E = void>
01798     struct gt_helper {};
01799
01800     template<typename v1, typename v2>
01801     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)> {
01802         using type = std::true_type;
01803     };
01804
01805     template<typename v1, typename v2>
01806     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)> {
01807         using type = std::false_type;
01808     };

```

```

01809
01810     template<typename v1, typename v2>
01811     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)>> {
01812         using type = std::false_type;
01813     };
01814
01815     // when high power is zero : strip
01816     template<typename P>
01817     struct simplify<P, std::enable_if_t<
01818         std::is_same<
01819             typename Ring::zero,
01820             typename P::aN
01821         >::value && (P::degree > 0)
01822     >> {
01823         using type = typename simplify<typename P::strip>::type;
01824     };
01825
01826     // otherwise : do nothing
01827     template<typename P>
01828     struct simplify<P, std::enable_if_t<
01829         !std::is_same<
01830             typename Ring::zero,
01831             typename P::aN
01832         >::value && (P::degree > 0)
01833     >> {
01834         using type = P;
01835     };
01836
01837     // do not simplify constants
01838     template<typename P>
01839     struct simplify<P, std::enable_if_t<P::degree == 0>> {
01840         using type = P;
01841     };
01842
01843     // addition at
01844     template<typename P1, typename P2, size_t index>
01845     struct add_at {
01846         using type =
01847             typename Ring::template add_t<
01848                 typename P1::template coeff_at_t<index>,
01849                 typename P2::template coeff_at_t<index>;
01850     };
01851
01852     template<typename P1, typename P2, size_t index>
01853     using add_at_t = typename add_at<P1, P2, index>::type;
01854
01855     template<typename P1, typename P2, std::size_t... I>
01856     struct add_low<P1, P2, std::index_sequence<I...> {
01857         using type = val<add_at_t<P1, P2, I>...>;
01858     };
01859
01860     // subtraction at
01861     template<typename P1, typename P2, size_t index>
01862     struct sub_at {
01863         using type =
01864             typename Ring::template sub_t<
01865                 typename P1::template coeff_at_t<index>,
01866                 typename P2::template coeff_at_t<index>;
01867     };
01868
01869     template<typename P1, typename P2, size_t index>
01870     using sub_at_t = typename sub_at<P1, P2, index>::type;
01871
01872     template<typename P1, typename P2, std::size_t... I>
01873     struct sub_low<P1, P2, std::index_sequence<I...> {
01874         using type = val<sub_at_t<P1, P2, I>...>;
01875     };
01876
01877     template<typename P1, typename P2>
01878     struct sub {
01879         using type = typename simplify<typename sub_low<
01880             P1,
01881             P2,
01882             internal::make_index_sequence_reverse<
01883                 std::max(P1::degree, P2::degree) + 1
01884             >::type>::type;
01885     };
01886
01887     // multiplication at
01888     template<typename v1, typename v2, size_t k, size_t index, size_t stop>
01889     struct mul_at_loop_helper {
01890         using type = typename Ring::template add_t<
01891             typename Ring::template mul_t<
01892                 typename v1::template coeff_at_t<index>,
01893                 typename v2::template coeff_at_t<k - index>
01894             >,
01895             typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type

```

```

01896         >;
01897     };
01898
01899     template<typename v1, typename v2, size_t k, size_t stop>
01900     struct mul_at_loop_helper<v1, v2, k, stop, stop> {
01901         using type = typename Ring::template mul_t<
01902             typename v1::template coeff_at_t<stop>,
01903             typename v2::template coeff_at_t<0>;
01904     };
01905
01906     template<typename v1, typename v2, size_t k, typename E = void>
01907     struct mul_at {};
01908
01909     template<typename v1, typename v2, size_t k>
01910     struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)>> {
01911         using type = typename Ring::zero;
01912     };
01913
01914     template<typename v1, typename v2, size_t k>
01915     struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)>> {
01916         using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01917     };
01918
01919     template<typename P1, typename P2, size_t index>
01920     using mul_at_t = typename mul_at<P1, P2, index>::type;
01921
01922     template<typename P1, typename P2, std::size_t... I>
01923     struct mul_low<P1, P2, std::index_sequence<I...> {
01924         using type = val<mul_at_t<P1, P2, I>...>;
01925     };
01926
01927     // division helper
01928     template<typename A, typename B, typename Q, typename R, typename E = void>
01929     struct div_helper {};
01930
01931     template<typename A, typename B, typename Q, typename R>
01932     struct div_helper<A, B, Q, R, std::enable_if_t<
01933         (R::degree < B::degree) ||
01934         (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)>> {
01935         using q_type = Q;
01936         using mod_type = R;
01937         using gcd_type = B;
01938     };
01939
01940     template<typename A, typename B, typename Q, typename R>
01941     struct div_helper<A, B, Q, R, std::enable_if_t<
01942         (R::degree >= B::degree) &&
01943         !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)>> {
01944     private: // NOLINT
01945         using rN = typename R::aN;
01946         using bN = typename B::aN;
01947         using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
01948             B::degree>::type;
01949         using rr = typename sub<R, typename mul<pT, B>::type>::type;
01950         using qq = typename add<Q, pT>::type;
01951     public:
01952         using q_type = typename div_helper<A, B, qq, rr>::q_type;
01953         using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
01954         using gcd_type = rr;
01955     };
01956
01957     template<typename A, typename B>
01958     struct div {
01959         static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
01960         using q_type = typename div_helper<A, B, zero, A>::q_type;
01961         using m_type = typename div_helper<A, B, zero, A>::mod_type;
01962     };
01963
01964     template<typename P>
01965     struct make_unit {
01966         using type = typename div<P, val<typename P::aN>::q_type;
01967     };
01968
01969     template<typename coeff, size_t deg>
01970     struct monomial {
01971         using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
01972     };
01973
01974     template<typename coeff>
01975     struct monomial<coeff, 0> {
01976         using type = val<coeff>;
01977     };
01978
01979     template<typename arithmeticType, typename P>
01980     struct horner_evaluation {
01981         template<size_t index, size_t stop>

```

```

01982         struct inner {
01983             static constexpr DEVICE INLINED arithmeticType func(
01984                 const arithmeticType& accum, const arithmeticType& x) {
01985                 return horner_evaluation<arithmeticType, P>::template inner<index + 1,
stop>::func(
01986                     internal::fma_helper<arithmeticType>::eval(
01987                         x,
01988                         accum,
01989                         P::template coeff_at_t<P::degree - index>::template
get<arithmeticType>(), x);
01990                     }
01991             };
01992
01993             template<size_t stop>
01994             struct inner<stop, stop> {
01995                 static constexpr DEVICE INLINED arithmeticType func(
01996                     const arithmeticType& accum, const arithmeticType& x) {
01997                     return accum;
01998                 }
01999             };
02000         };
02001
02002         template<typename coeff, typename... coeffs>
02003         struct string_helper {
02004             static std::string func() {
02005                 std::string tail = string_helper<coeffs...>::func();
02006                 std::string result = "";
02007                 if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
02008                     return tail;
02009                 } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
02010                     if (sizeof...(coeffs) == 1) {
02011                         result += "x";
02012                     } else {
02013                         result += "x^" + std::to_string(sizeof...(coeffs));
02014                     }
02015                 } else {
02016                     if (sizeof...(coeffs) == 1) {
02017                         result += coeff::to_string() + " x";
02018                     } else {
02019                         result += coeff::to_string()
02020                             + " x^" + std::to_string(sizeof...(coeffs));
02021                     }
02022                 }
02023
02024                 if (!tail.empty()) {
02025                     if (tail.at(0) != '-') {
02026                         result += " + " + tail;
02027                     } else {
02028                         result += " - " + tail.substr(1);
02029                     }
02030                 }
02031                 return result;
02032             }
02033         };
02034
02035         template<typename coeff>
02036         struct string_helper<coeff> {
02037             static std::string func() {
02038                 if (!std::is_same<coeff, typename Ring::zero>::value) {
02039                     return coeff::to_string();
02040                 } else {
02041                     return "";
02042                 }
02043             }
02044         };
02045     };
02046
02047     public:
02048         template<typename P>
02049         using simplify_t = typename simplify<P>::type;
02050
02051         template<typename v1, typename v2>
02052         using add_t = typename add<v1, v2>::type;
02053
02054         template<typename v1, typename v2>
02055         using sub_t = typename sub<v1, v2>::type;
02056
02057         template<typename v1, typename v2>
02058         using mul_t = typename mul<v1, v2>::type;
02059
02060         template<typename v1, typename v2>
02061         using eq_t = typename eq_helper<v1, v2>::type;
02062
02063         template<typename v1, typename v2>
02064         using lt_t = typename lt_helper<v1, v2>::type;
02065
02066         template<typename v1, typename v2>

```



```

02087     using gt_t = typename gt_helper<v1, v2>::type;
02088
02092     template<typename v1, typename v2>
02093     using div_t = typename div<v1, v2>::q_type;
02094
02098     template<typename v1, typename v2>
02099     using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
02100
02104     template<typename coeff, size_t deg>
02105     using monomial_t = typename monomial<coeff, deg>::type;
02106
02109     template<typename v>
02110     using derive_t = typename derive_helper<v>::type;
02111
02114     template<typename v>
02115     using pos_t = typename Ring::template pos_t<typename v::aN>;
02116
02119     template<typename v>
02120     static constexpr bool pos_v = pos_t<v>::value;
02121
02125     template<typename v1, typename v2>
02126     using gcd_t = std::conditional_t<
02127         Ring::is_euclidean_domain,
02128         typename make_unit<gcd_t<polynomial<Ring>, v1, v2>::type,
02129         void>;
02130
02133     template<auto x>
02134     using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
02135
02138     template<typename v>
02139     using inject_ring_t = val<v>;
02140 };
02141 } // namespace aerobus
02142
02143 // fraction field
02144 namespace aerobus {
02145     namespace internal {
02146         template<typename Ring, typename E = void>
02147         requires IsEuclideanDomain<Ring>
02148         struct _FractionField {};
02149
02150         template<typename Ring>
02151         requires IsEuclideanDomain<Ring>
02152         struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain>> {
02153             static constexpr bool is_field = true;
02154             static constexpr bool is_euclidean_domain = true;
02155
02156         private:
02157             template<typename val1, typename val2, typename E = void>
02158             struct to_string_helper {};
02159
02160             template<typename val1, typename val2>
02161             struct to_string_helper<val1, val2,
02162                 std::enable_if_t<
02163                     Ring::template eq_t<
02164                         val2, typename Ring::one
02165                         >::value
02166                     >
02167             > {
02168                 static std::string func() {
02169                     return val1::to_string();
02170                 }
02171             };
02172
02173             template<typename val1, typename val2>
02174             struct to_string_helper<val1, val2,
02175                 std::enable_if_t<
02176                     !Ring::template eq_t<
02177                         val2,
02178                         typename Ring::one
02179                         >::value
02180                     >
02181             > {
02182                 static std::string func() {
02183                     return "(" + val1::to_string() + " ) / ( " + val2::to_string() + " )";
02184                 }
02185             };
02186
02187         public:
02188             template<typename val1, typename val2>
02189             struct val {
02190                 using x = val1;
02191                 using y = val2;
02192                 using is_zero_t = typename val1::is_zero_t;
02193                 static constexpr bool is_zero_v = val1::is_zero_t::value;
02194
02195                 using ring_type = Ring;

```

```

02205         using enclosing_type = _FractionField<Ring>;
02206
02209         static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
02210
02214         template<typename valueType>
02215         static constexpr INLINED_DEVICE valueType get() {
02216             return internal::staticcast<valueType, typename ring_type::inner_type>::template
func<x::v>() /
02217             internal::staticcast<valueType, typename ring_type::inner_type>::template
func<y::v>();
02218     }
02219
02222     static std::string to_string() {
02223         return to_string_helper<val1, val2>::func();
02224     }
02225
02230     template<typename arithmeticType>
02231     static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& v) {
02232         return x::eval(v) / y::eval(v);
02233     }
02234 };
02235
02237 using zero = val<typename Ring::zero, typename Ring::one>;
02239 using one = val<typename Ring::one, typename Ring::one>;
02240
02243 template<typename v>
02244 using inject_t = val<v, typename Ring::one>;
02245
02248 template<auto x>
02249 using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
Ring::one>;
02250
02253 template<typename v>
02254 using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
02255
02257 using ring_type = Ring;
02258
02259 private:
02260     template<typename v, typename E = void>
02261     struct simplify {};
02262
02263     // x = 0
02264     template<typename v>
02265     struct simplify<v, std::enable_if_t<v::x::is_zero_t::value> {
02266         using type = typename _FractionField<Ring>::zero;
02267     };
02268
02269     // x != 0
02270     template<typename v>
02271     struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value> {
02272     private:
02273         using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
02274         using newx = typename Ring::template div_t<typename v::x, _gcd>;
02275         using newy = typename Ring::template div_t<typename v::y, _gcd>;
02276
02277         using posx = std::conditional_t<
02278             !Ring::template pos_v<newx>,
02279             typename Ring::template sub_t<typename Ring::zero, newx>,
02280             newx>;
02281         using posy = std::conditional_t<
02282             !Ring::template pos_v<newy>,
02283             typename Ring::template sub_t<typename Ring::zero, newy>,
02284             newy>;
02285     public:
02286         using type = typename _FractionField<Ring>::template val<posx, posy>;
02287     };
02288
02289 public:
02292     template<typename v>
02293     using simplify_t = typename simplify<v>::type;
02294
02295 private:
02296     template<typename v1, typename v2>
02297     struct add {
02298     private:
02299         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02300         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02301         using dividend = typename Ring::template add_t<a, b>;
02302         using divider = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02303         using g = typename Ring::template gcd_t<dividend, divider>;
02304
02305     public:
02306         using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
diviser>;
02307     };
02308
02309     template<typename v>

```

```

02310 struct pos {
02311     using type = std::conditional_t<
02312         (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
02313         (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
02314         std::true_type,
02315         std::false_type>;
02316 };
02317
02318 template<typename v1, typename v2>
02319 struct sub {
02320     private:
02321         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02322         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02323         using dividend = typename Ring::template sub_t<a, b>;
02324         using divisor = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02325         using g = typename Ring::template gcd_t<dividend, divisor>;
02326
02327     public:
02328         using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
diviser>;
02329     };
02330
02331 template<typename v1, typename v2>
02332 struct mul {
02333     private:
02334         using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
02335         using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02336
02337     public:
02338         using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
02339     };
02340
02341 template<typename v1, typename v2, typename E = void>
02342 struct div {};
02343
02344 template<typename v1, typename v2>
02345 struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
_FractionField<Ring>::zero>::value> {
02346     private:
02347         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02348         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02349
02350     public:
02351         using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
02352     };
02353
02354 template<typename v1, typename v2>
02355 struct div<v1, v2, std::enable_if_t<
std::is_same<zero, v1>::value && std::is_same<v2, zero>::value> {
02356     using type = one;
02357 };
02358
02359 template<typename v1, typename v2>
02360 struct eq {
02361     using type = std::conditional_t<
02362         std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
02363         std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
02364         std::true_type,
02365         std::false_type>;
02366 };
02367
02368 template<typename v1, typename v2, typename E = void>
02369 struct gt;
02370
02371 template<typename v1, typename v2>
02372 struct gt<v1, v2, std::enable_if_t<
02373     (eq<v1, v2>::type::value)
02374     >> {
02375     using type = std::false_type;
02376 };
02377
02378 template<typename v1, typename v2>
02379 struct gt<v1, v2, std::enable_if_t<
02380     (!eq<v1, v2>::type::value) &&
02381     (!pos<v1>::type::value) && (!pos<v2>::type::value)
02382     >> {
02383     using type = typename gt<
02384         typename sub<zero, v1>::type, typename sub<zero, v2>::type
02385     >::type;
02386 };
02387
02388 template<typename v1, typename v2>
02389 struct gt<v1, v2, std::enable_if_t<
02390     (!eq<v1, v2>::type::value) &&
02391     (pos<v1>::type::value) && (!pos<v2>::type::value)
02392     >> {
02393     using type = std::true_type;
02394

```

```

02395         };
02396
02397     template<typename v1, typename v2>
02398     struct gt<v1, v2, std::enable_if_t<
02399         (!eq<v1, v2>::type::value) &&
02400         (!pos<v1>::type::value) && (pos<v2>::type::value)
02401         >> {
02402         using type = std::false_type;
02403     };
02404
02405     template<typename v1, typename v2>
02406     struct gt<v1, v2, std::enable_if_t<
02407         (!eq<v1, v2>::type::value) &&
02408         (pos<v1>::type::value) && (pos<v2>::type::value)
02409         >> {
02410         using type = typename Ring::template gt_t<
02411             typename Ring::template mul_t<v1::x, v2::y>,
02412             typename Ring::template mul_t<v2::y, v2::x>
02413         >;
02414     };
02415
02416     public:
02417     template<typename v1, typename v2>
02418     using add_t = typename add<v1, v2>::type;
02419
02420     template<typename v1, typename v2>
02421     using mod_t = zero;
02422
02423     template<typename v1, typename v2>
02424     using gcd_t = v1;
02425
02426     template<typename v1, typename v2>
02427     using sub_t = typename sub<v1, v2>::type;
02428
02429     template<typename v1, typename v2>
02430     using mul_t = typename mul<v1, v2>::type;
02431
02432     template<typename v1, typename v2>
02433     using div_t = typename div<v1, v2>::type;
02434
02435     template<typename v1, typename v2>
02436     using eq_t = typename eq<v1, v2>::type;
02437
02438     template<typename v1, typename v2>
02439     static constexpr bool eq_v = eq<v1, v2>::type::value;
02440
02441     template<typename v1, typename v2>
02442     using gt_t = typename gt<v1, v2>::type;
02443
02444     template<typename v1, typename v2>
02445     static constexpr bool gt_v = gt<v1, v2>::type::value;
02446
02447     template<typename v1>
02448     using pos_t = typename pos<v1>::type;
02449
02450     template<typename v>
02451     static constexpr bool pos_v = pos<t<v>>::value;
02452 };
02453
02454 template<typename Ring, typename E = void>
02455 requires IsEuclideanDomain<Ring>
02456 struct FractionFieldImpl {};
02457
02458 // fraction field of a field is the field itself
02459 template<typename Field>
02460 requires IsEuclideanDomain<Field>
02461 struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field> {
02462     using type = Field;
02463     template<typename v>
02464     using inject_t = v;
02465 };
02466
02467 // fraction field of a ring is the actual fraction field
02468 template<typename Ring>
02469 requires IsEuclideanDomain<Ring>
02470 struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field> {
02471     using type = _FractionField<Ring>;
02472 };
02473 } // namespace internal
02474
02475 template<typename Ring>
02476 requires IsEuclideanDomain<Ring>
02477 using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02478
02479 template<typename Ring>
02480 struct Embed<Ring, FractionField<Ring> {
02481     template<typename v>

```

```

02524         using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02525     };
02526 } // namespace aerobus
02527
02528
02529 // short names for common types
02530 namespace aerobus {
02531     template<typename X, typename Y>
02532     requires IsRing<typename X::enclosing_type> &&
02533     (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02534     using add_t = typename X::enclosing_type::template add_t<X, Y>;
02535
02536     template<typename X, typename Y>
02537     requires IsRing<typename X::enclosing_type> &&
02538     (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02539     using sub_t = typename X::enclosing_type::template sub_t<X, Y>;
02540
02541     template<typename X, typename Y>
02542     requires IsRing<typename X::enclosing_type> &&
02543     (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02544     using mul_t = typename X::enclosing_type::template mul_t<X, Y>;
02545
02546     template<typename X, typename Y>
02547     requires IsRing<typename X::enclosing_type> &&
02548     (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02549     using div_t = typename X::enclosing_type::template div_t<X, Y>;
02550
02551     using q32 = FractionField<i32>;
02552
02553     using fpq32 = FractionField<polynomial<q32>>;
02554
02555     using q64 = FractionField<i64>;
02556
02557     using pi64 = polynomial<i64>;
02558
02559     using pq64 = polynomial<q64>;
02560
02561     using fpq64 = FractionField<polynomial<q64>>;
02562
02563     template<typename Ring, typename v1, typename v2>
02564     using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02565
02566     template<typename v>
02567     using embed_int_poly_in_fractions_t =
02568         typename Embed<
02569             polynomial<typename v::ring_type>,
02570             polynomial<FractionField<typename v::ring_type>>::template type<v>;
02571
02572     template<int64_t p, int64_t q>
02573     using make_q64_t = typename q64::template simplify_t<
02574         typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>>;
02575
02576     template<int32_t p, int32_t q>
02577     using make_q32_t = typename q32::template simplify_t<
02578         typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>>;
02579
02580     template<typename Ring, typename v1, typename v2>
02581     using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02582     template<typename Ring, typename v1, typename v2>
02583     using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02584
02585     template<>
02586     struct Embed<q32, q64> {
02587         template<typename v>
02588         using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02589     };
02590
02591     template<typename Small, typename Large>
02592     struct Embed<polynomial<Small>, polynomial<Large>> {
02593     private:
02594         template<typename v, typename i>
02595         struct at_low;
02596
02597         template<typename v, size_t i>
02598         struct at_index {
02599             using type = typename Embed<Small, Large>::template
02600                 type<typename v::template coeff_at_t<i>>;
02601         };
02602
02603         template<typename v, size_t... Is>
02604         struct at_low<v, std::index_sequence<Is...>> {
02605             using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
02606         };
02607
02608     public:
02609         template<typename v>
02610         using type = typename at_low<v, typename internal::make_index_sequence_reverse<v::degree +

```

```

1>::type;
02663     };
02664
02665     template<typename Ring, auto... xs>
02666     using make_int_polynomial_t = typename polynomial<Ring>::template val<
02670         typename Ring::template inject_constant_t<xs>...>;
02671
02672     template<typename Ring, auto... xs>
02673     using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<
02674         typename FractionField<Ring>::template inject_constant_t<xs>...>;
02675 } // namespace aerobus
02676
02677 // Taylor series and common integers (factorial, bernoulli...) appearing in Taylor coefficients
02678 namespace aerobus {
02679     namespace internal {
02680         template<typename T, size_t x, typename E = void>
02681         struct factorial {};
02682
02683         template<typename T, size_t x>
02684         struct factorial<T, x, std::enable_if_t<(x > 0)>> {
02685             private:
02686                 template<typename, size_t, typename>
02687                 friend struct factorial;
02688             public:
02689                 using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
02690 x - 1>::type>;
02691         static constexpr typename T::inner_type value = type::template get<typename
02692 T::inner_type>();
02693     };
02694
02695     template<typename T>
02696     struct factorial<T, 0> {
02697     public:
02698         using type = typename T::one;
02699         static constexpr typename T::inner_type value = type::template get<typename
02700 T::inner_type>();
02701     };
02702 } // namespace internal
02703
02704 template<typename T, size_t i>
02705 using factorial_t = typename internal::factorial<T, i>::type;
02706
02707 template<typename T, size_t i>
02708 inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02709
02710 namespace internal {
02711     template<typename T, size_t k, size_t n, typename E = void>
02712     struct combination_helper {};
02713
02714     template<typename T, size_t k, size_t n>
02715     struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)>> {
02716         using type = typename FractionField<T>::template mul_t<
02717             typename combination_helper<T, k - 1, n - 1>::type,
02718             makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02719     };
02720
02721     template<typename T, size_t k, size_t n>
02722     struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)>> {
02723         using type = typename combination_helper<T, n - k, n>::type;
02724     };
02725
02726     template<typename T, size_t n>
02727     struct combination_helper<T, 0, n> {
02728         using type = typename FractionField<T>::one;
02729     };
02730
02731     template<typename T, size_t k, size_t n>
02732     struct combination {
02733         using type = typename internal::combination_helper<T, k, n>::type::x;
02734         static constexpr typename T::inner_type value =
02735             internal::combination_helper<T, k, n>::type::template get<typename
02736 T::inner_type>();
02737     };
02738 } // namespace internal
02739
02740 template<typename T, size_t k, size_t n>
02741 using combination_t = typename internal::combination<T, k, n>::type;
02742
02743 template<typename T, size_t k, size_t n>
02744 inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02745
02746 namespace internal {
02747     template<typename T, size_t m>
02748     struct bernoulli;
02749
02750     template<typename T, typename accum, size_t k, size_t m>
02751     struct bernoulli_helper {

```

```

02763         using type = typename bernoulli_helper<
02764             T,
02765             addfractions_t<T,
02766                 accum,
02767                 mulfractions_t<T,
02768                     makefraction_t<T,
02769                         combination_t<T, k, m + 1>,
02770                         typename T::one>,
02771                         typename bernoulli<T, k>::type
02772                     >,
02773                     >,
02774                     k + 1,
02775                     m>::type;
02776     };
02777
02778     template<typename T, typename accum, size_t m>
02779     struct bernoulli_helper<T, accum, m, m> {
02780         using type = accum;
02781     };
02782
02783
02784
02785     template<typename T, size_t m>
02786     struct bernoulli {
02787         using type = typename FractionField<T>::template mul_t<
02788             typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02789             makefraction_t<T,
02790                 typename T::template val<static_cast<typename T::inner_type>(-1)>,
02791                 typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02792             >
02793         >;
02794
02795         template<typename floatType>
02796         static constexpr floatType value = type::template get<floatType>();
02797     };
02798
02799     template<typename T>
02800     struct bernoulli<T, 0> {
02801         using type = typename FractionField<T>::one;
02802
02803         template<typename floatType>
02804         static constexpr floatType value = type::template get<floatType>();
02805     };
02806 } // namespace internal
02807
02811 template<typename T, size_t n>
02812 using bernoulli_t = typename internal::bernoulli<T, n>::type;
02813
02818 template<typename FloatType, typename T, size_t n>
02819 inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02820
02821 // bell numbers
02822 namespace internal {
02823     template<typename T, size_t n, typename E = void>
02824     struct bell_helper;
02825
02826     template<typename T, size_t n>
02827     struct bell_helper<T, n, std::enable_if_t<(n > 1)>> {
02828         template<typename accum, size_t i, size_t stop>
02829         struct sum_helper {
02830             private:
02831                 using left = typename T::template mul_t<
02832                     combination_t<T, i, n-1>,
02833                     typename bell_helper<T, i>::type>;
02834                 using new_accum = typename T::template add_t<accum, left>;
02835             public:
02836                 using type = typename sum_helper<new_accum, i+1, stop>::type;
02837         };
02838
02839         template<typename accum, size_t stop>
02840         struct sum_helper<accum, stop, stop> {
02841             using type = accum;
02842         };
02843
02844         using type = typename sum_helper<typename T::zero, 0, n>::type;
02845     };
02846
02847     template<typename T>
02848     struct bell_helper<T, 0> {
02849         using type = typename T::one;
02850     };
02851
02852     template<typename T>
02853     struct bell_helper<T, 1> {
02854         using type = typename T::one;
02855     };
02856 } // namespace internal

```

```

02857
02861     template<typename T, size_t n>
02862     using bell_t = typename internal::bell_helper<T, n>::type;
02863
02867     template<typename T, size_t n>
02868     static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
02869
02870     namespace internal {
02871         template<typename T, int k, typename E = void>
02872         struct alternate {};
02873
02874         template<typename T, int k>
02875         struct alternate<T, k, std::enable_if_t<k % 2 == 0> {
02876             using type = typename T::one;
02877             static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02878         };
02879
02880         template<typename T, int k>
02881         struct alternate<T, k, std::enable_if_t<k % 2 != 0> {
02882             using type = typename T::template sub_t<typename T::zero, typename T::one>;
02883             static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02884         };
02885     } // namespace internal
02886
02889     template<typename T, int k>
02890     using alternate_t = typename internal::alternate<T, k>::type;
02891
02892     namespace internal {
02893         template<typename T, int n, int k, typename E = void>
02894         struct stirling_helper {};
02895
02896         template<typename T>
02897         struct stirling_helper<T, 0, 0> {
02898             using type = typename T::one;
02899         };
02900
02901         template<typename T, int n>
02902         struct stirling_helper<T, n, 0, std::enable_if_t<(n > 0)> {
02903             using type = typename T::zero;
02904         };
02905
02906         template<typename T, int n>
02907         struct stirling_helper<T, 0, n, std::enable_if_t<(n > 0)> {
02908             using type = typename T::zero;
02909         };
02910
02911         template<typename T, int n, int k>
02912         struct stirling_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)> {
02913             using type = typename T::template sub_t<
02914                 typename stirling_helper<T, n-1, k-1>::type,
02915                 typename T::template mul_t<
02916                     typename T::template inject_constant_t<n-1>,
02917                     typename stirling_helper<T, n-1, k>::type
02918                 >;
02919         };
02920     } // namespace internal
02921
02926     template<typename T, int n, int k>
02927     using stirling_signed_t = typename internal::stirling_helper<T, n, k>::type;
02928
02933     template<typename T, int n, int k>
02934     using stirling_unsigned_t = abs_t<typename internal::stirling_helper<T, n, k>::type>;
02935
02940     template<typename T, int n, int k>
02941     static constexpr typename T::inner_type stirling_signed_v = stirling_signed_t<T, n, k>::v;
02942
02943
02948     template<typename T, int n, int k>
02949     static constexpr typename T::inner_type stirling_unsigned_v = stirling_unsigned_t<T, n, k>::v;
02950
02953     template<typename T, size_t k>
02954     inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
02955
02956     namespace internal {
02957         template<typename T>
02958         struct pow_scalar {
02959             template<size_t p>
02960             static constexpr DEVICE INLINED T func(const T& x) { return p == 0 ? static_cast<T>(1) :
02961                 p % 2 == 0 ? func<p/2>(x) * func<p/2>(x) :
02962                 x * func<p/2>(x) * func<p/2>(x);
02963             }
02964         };
02965
02966         template<typename T, typename p, size_t n, typename E = void>
02967         requires IsEuclideanDomain<T>

```



```

02968     struct pow;
02969
02970     template<typename T, typename p, size_t n>
02971     struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)> {
02972         using type = typename T::template mul_t<
02973             typename pow<T, p, n/2>::type,
02974             typename pow<T, p, n/2>::type
02975         >;
02976     };
02977
02978     template<typename T, typename p, size_t n>
02979     struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)> {
02980         using type = typename T::template mul_t<
02981             p,
02982             typename T::template mul_t<
02983                 typename pow<T, p, n/2>::type,
02984                 typename pow<T, p, n/2>::type
02985             >
02986         >;
02987     };
02988
02989     template<typename T, typename p, size_t n>
02990     struct pow<T, p, n, std::enable_if_t<n == 0> { using type = typename T::one; };
02991 } // namespace internal
02992
02993 template<typename T, typename p, size_t n>
02994 using pow_t = typename internal::pow<T, p, n>::type;
02995
02996 template<typename T, typename p, size_t n>
02997 static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
02998
02999 template<typename T, size_t p>
03000 static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
03001     internal::pow_scalar<T>::template func<p>(x); }
03002
03003 namespace internal {
03004     template<typename, template<typename, size_t> typename, class>
03005     struct make_taylor_impl;
03006
03007     template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
03008     struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...> {
03009         using type = typename polynomial<FractionField<T>::template val<typename coeff_at<T,
03010             Is>::type...>;
03011     };
03012 }
03013
03014 template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
03015 using taylor = typename internal::make_taylor_impl<
03016     T,
03017     coeff_at,
03018     internal::make_index_sequence_reverse<deg + 1>::type;
03019
03020 namespace internal {
03021     template<typename T, size_t i>
03022     struct exp_coeff {
03023         using type = makefraction_t<T, typename T::one, factorial_t<T, i>;
03024     };
03025
03026     template<typename T, size_t i, typename E = void>
03027     struct sin_coeff_helper {};
03028
03029     template<typename T, size_t i>
03030     struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03031         using type = typename FractionField<T>::zero;
03032     };
03033
03034     template<typename T, size_t i>
03035     struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03036         using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>;
03037     };
03038
03039     template<typename T, size_t i>
03040     struct sin_coeff {
03041         using type = typename sin_coeff_helper<T, i>::type;
03042     };
03043
03044     template<typename T, size_t i, typename E = void>
03045     struct sh_coeff_helper {};
03046
03047     template<typename T, size_t i>
03048     struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03049         using type = typename FractionField<T>::zero;
03050     };
03051
03052     template<typename T, size_t i>
03053     struct sh_coeff {
03054         using type = makefraction_t<T, typename T::one, factorial_t<T, i>;
03055     };
03056 }

```

```

03065     };
03066
03067     template<typename T, size_t i>
03068     struct sh_coeff {
03069         using type = typename sh_coeff_helper<T, i>::type;
03070     };
03071
03072     template<typename T, size_t i, typename E = void>
03073     struct cos_coeff_helper {};
03074
03075     template<typename T, size_t i>
03076     struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03077         using type = typename FractionField<T>::zero;
03078     };
03079
03080     template<typename T, size_t i>
03081     struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03082         using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03083     };
03084
03085     template<typename T, size_t i>
03086     struct cos_coeff {
03087         using type = typename cos_coeff_helper<T, i>::type;
03088     };
03089
03090     template<typename T, size_t i, typename E = void>
03091     struct cosh_coeff_helper {};
03092
03093     template<typename T, size_t i>
03094     struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03095         using type = typename FractionField<T>::zero;
03096     };
03097
03098     template<typename T, size_t i>
03099     struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03100         using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03101     };
03102
03103     template<typename T, size_t i>
03104     struct cosh_coeff {
03105         using type = typename cosh_coeff_helper<T, i>::type;
03106     };
03107
03108     template<typename T, size_t i>
03109     struct geom_coeff { using type = typename FractionField<T>::one; };
03110
03111
03112     template<typename T, size_t i, typename E = void>
03113     struct atan_coeff_helper;
03114
03115     template<typename T, size_t i>
03116     struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03117         using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>>;
03118     };
03119
03120     template<typename T, size_t i>
03121     struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03122         using type = typename FractionField<T>::zero;
03123     };
03124
03125     template<typename T, size_t i>
03126     struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
03127
03128     template<typename T, size_t i, typename E = void>
03129     struct asin_coeff_helper;
03130
03131     template<typename T, size_t i>
03132     struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03133         using type = makefraction_t<T,
03134             factorial_t<T, i - 1>,
03135             typename T::template mul_t<
03136                 typename T::template val<i>,
03137                 T::template mul_t<
03138                     pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
03139                     pow<T, factorial_t<T, i / 2>, 2
03140                 >
03141             >
03142         >>;
03143     };
03144
03145     template<typename T, size_t i>
03146     struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03147         using type = typename FractionField<T>::zero;
03148     };
03149
03150     template<typename T, size_t i>
03151     struct asin_coeff {

```

```

03152         using type = typename asin_coeff_helper<T, i>::type;
03153     };
03154
03155     template<typename T, size_t i>
03156     struct lnpl_coeff {
03157         using type = makefraction_t<T,
03158             alternate_t<T, i + 1>,
03159             typename T::template val<i>;
03160     };
03161
03162     template<typename T>
03163     struct lnpl_coeff<T, 0> { using type = typename FractionField<T>::zero; };
03164
03165     template<typename T, size_t i, typename E = void>
03166     struct asinh_coeff_helper;
03167
03168     template<typename T, size_t i>
03169     struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03170         using type = makefraction_t<T,
03171             typename T::template mul_t<
03172                 alternate_t<T, i / 2>,
03173                 factorial_t<T, i - 1>
03174             >,
03175             typename T::template mul_t<
03176                 typename T::template mul_t<
03177                     typename T::template val<i>,
03178                     pow_t<T, factorial_t<T, i / 2>, 2>
03179                 >,
03180                 pow_t<T, typename T::template inject_constant_t<4>, i / 2>
03181             >
03182         >;
03183     };
03184
03185     template<typename T, size_t i>
03186     struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03187         using type = typename FractionField<T>::zero;
03188     };
03189
03190     template<typename T, size_t i>
03191     struct asinh_coeff {
03192         using type = typename asinh_coeff_helper<T, i>::type;
03193     };
03194
03195     template<typename T, size_t i, typename E = void>
03196     struct atanh_coeff_helper;
03197
03198     template<typename T, size_t i>
03199     struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03200         // 1/i
03201         using type = typename FractionField<T>::template val<
03202             typename T::one,
03203             typename T::template inject_constant_t<i>;
03204     };
03205
03206     template<typename T, size_t i>
03207     struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03208         using type = typename FractionField<T>::zero;
03209     };
03210
03211     template<typename T, size_t i>
03212     struct atanh_coeff {
03213         using type = typename atanh_coeff_helper<T, i>::type;
03214     };
03215
03216     template<typename T, size_t i, typename E = void>
03217     struct tan_coeff_helper;
03218
03219     template<typename T, size_t i>
03220     struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0> {
03221         using type = typename FractionField<T>::zero;
03222     };
03223
03224     template<typename T, size_t i>
03225     struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0> {
03226     private:
03227         // 4^((i+1)/2)
03228         using _4p = typename FractionField<T>::template inject_t<
03229             pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2>;
03230         // 4^((i+1)/2) - 1
03231         using _4pml = typename FractionField<T>::template
03232             sub_t<_4p, typename FractionField<T>::one>;
03233         // (-1)^((i-1)/2)
03234         using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2>;
03235         using dividend = typename FractionField<T>::template mul_t<
03236             altp,
03237             FractionField<T>::template mul_t<
03238                 _4p,

```

```

03238         FractionField<T>::template mul_t<
03239             _4pml,
03240             bernoulli_t<T, (i + 1)>
03241         >
03242     >
03243 >;
03244 public:
03245     using type = typename FractionField<T>::template div_t<dividend,
03246         typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03247 >;
03248
03249 template<typename T, size_t i>
03250 struct tan_coeff {
03251     using type = typename tan_coeff_helper<T, i>::type;
03252 };
03253
03254 template<typename T, size_t i, typename E = void>
03255 struct tanh_coeff_helper;
03256
03257 template<typename T, size_t i>
03258 struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0> {
03259     using type = typename FractionField<T>::zero;
03260 };
03261
03262 template<typename T, size_t i>
03263 struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0> {
03264 private:
03265     using _4p = typename FractionField<T>::template inject_t<
03266         pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2>;
03267     using _4pml = typename FractionField<T>::template
03268         sub_t<_4p, typename FractionField<T>::one>;
03269     using dividend =
03270         typename FractionField<T>::template mul_t<
03271             _4p,
03272             typename FractionField<T>::template mul_t<
03273                 _4pml,
03274                 bernoulli_t<T, (i + 1)>>::type;
03275 public:
03276     using type = typename FractionField<T>::template div_t<dividend,
03277         FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03278 >;
03279
03280 template<typename T, size_t i>
03281 struct tanh_coeff {
03282     using type = typename tanh_coeff_helper<T, i>::type;
03283 };
03284 } // namespace internal
03285
03286 template<typename Integers, size_t deg>
03287 using exp = taylor<Integers, internal::exp_coeff, deg>;
03288
03289 template<typename Integers, size_t deg>
03290 using expml = typename polynomial<FractionField<Integers>>::template sub_t<
03291     exp<Integers, deg>,
03292     typename polynomial<FractionField<Integers>>::one>;
03293
03294 template<typename Integers, size_t deg>
03295 using lnpl = taylor<Integers, internal::lnpl_coeff, deg>;
03296
03297 template<typename Integers, size_t deg>
03298 using atan = taylor<Integers, internal::atan_coeff, deg>;
03299
03300 template<typename Integers, size_t deg>
03301 using sin = taylor<Integers, internal::sin_coeff, deg>;
03302
03303 template<typename Integers, size_t deg>
03304 using sinh = taylor<Integers, internal::sh_coeff, deg>;
03305
03306 template<typename Integers, size_t deg>
03307 using cosh = taylor<Integers, internal::cosh_coeff, deg>;
03308
03309 template<typename Integers, size_t deg>
03310 using cos = taylor<Integers, internal::cos_coeff, deg>;
03311
03312 template<typename Integers, size_t deg>
03313 using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
03314
03315 template<typename Integers, size_t deg>
03316 using asin = taylor<Integers, internal::asin_coeff, deg>;
03317
03318 template<typename Integers, size_t deg>
03319 using asinh = taylor<Integers, internal::asinh_coeff, deg>;
03320
03321 template<typename Integers, size_t deg>
03322 using atanh = taylor<Integers, internal::atanh_coeff, deg>;
03323
03324 template<typename Integers, size_t deg>

```

```

03370     using tan = taylor<Integers, internal::tan_coeff, deg>;
03371
03376     template<typename Integers, size_t deg>
03377     using tanh = taylor<Integers, internal::tanh_coeff, deg>;
03378 } // namespace aerobus
03379
03380 // continued fractions
03381 namespace aerobus {
03384     template<int64_t... values>
03385     struct ContinuedFraction {};
03386
03389     template<int64_t a0>
03390     struct ContinuedFraction<a0> {
03392         using type = typename q64::template inject_constant_t<a0>;
03394         static constexpr double val = static_cast<double>(a0);
03395     };
03396
03400     template<int64_t a0, int64_t... rest>
03401     struct ContinuedFraction<a0, rest...> {
03403         using type = q64::template add_t<
03404             typename q64::template inject_constant_t<a0>,
03405             typename q64::template div_t<
03406                 typename q64::one,
03407                 typename ContinuedFraction<rest...>::type
03408             >>;
03409
03411         static constexpr double val = type::template get<double>();
03412     };
03413
03417     using PI_fraction =
03418     ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1>;
03419     using E_fraction =
03420     ContinuedFraction<2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1>;
03421     using SQRT2_fraction =
03422     ContinuedFraction<1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2>;
03423     using SQRT3_fraction =
03424     ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2>;
03425 // NOLINT
03426 } // namespace aerobus
03427
03428 // known polynomials
03429 namespace aerobus {
03430     // CChebyshev
03431     namespace internal {
03432         template<int kind, size_t deg, typename I>
03433         struct chebyshev_helper {
03434             using type = typename polynomial<I>::template sub_t<
03435                 typename polynomial<I>::template mul_t<
03436                     typename polynomial<I>::template mul_t<
03437                         typename polynomial<I>::template inject_constant_t<2>,
03438                         typename polynomial<I>::X>,
03439                         typename chebyshev_helper<kind, deg - 1, I>::type
03440                     >,
03441                     typename chebyshev_helper<kind, deg - 2, I>::type
03442                 >;
03443         };
03444
03445         template<typename I>
03446         struct chebyshev_helper<1, 0, I> {
03447             using type = typename polynomial<I>::one;
03448         };
03449
03450         template<typename I>
03451         struct chebyshev_helper<1, 1, I> {
03452             using type = typename polynomial<I>::X;
03453         };
03454
03455         template<typename I>
03456         struct chebyshev_helper<2, 0, I> {
03457             using type = typename polynomial<I>::one;
03458         };
03459
03460         template<typename I>
03461         struct chebyshev_helper<2, 1, I> {
03462             using type = typename polynomial<I>::template mul_t<
03463                 typename polynomial<I>::template inject_constant_t<2>,
03464                 typename polynomial<I>::X>;
03465         };
03466     } // namespace internal
03467
03468     // Laguerre
03469     namespace internal {
03470         template<size_t deg, typename I>
03471         struct laguerre_helper {
03472             using Q = FractionField<I>;
03473             using PQ = polynomial<Q>;
03474         };
03475     }

```

```

03473     private:
03474         // Lk = (1 / k) * ((2 * k - 1 - x) * lkm1 - (k - 2)Lkm2)
03475         using lnm2 = typename laguerre_helper<deg - 2, I>::type;
03476         using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03477         // -x + 2k-1
03478         using p = typename PQ::template val<
03479             typename Q::template inject_constant_t<-1>,
03480             typename Q::template inject_constant_t<2 * deg - 1>;
03481         // 1/n
03482         using factor = typename PQ::template inject_ring_t<
03483             typename Q::template val<typename I::one, typename I::template
inject_constant_t<deg>>;
03484
03485     public:
03486         using type = typename PQ::template mul_t <
03487             factor,
03488             typename PQ::template sub_t<
03489                 typename PQ::template mul_t<
03490                     p,
03491                     lnm1
03492                 >,
03493                 typename PQ::template mul_t<
03494                     typename PQ::template inject_constant_t<deg-1>,
03495                     lnm2
03496                 >
03497             >
03498         >;
03499     };
03500
03501     template<typename I>
03502     struct laguerre_helper<0, I> {
03503         using type = typename polynomial<FractionField<I>::one;
03504     };
03505
03506     template<typename I>
03507     struct laguerre_helper<1, I> {
03508     private:
03509         using PQ = polynomial<FractionField<I>;
03510     public:
03511         using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
03512     };
03513 } // namespace internal
03514
03515 // Bernstein
03516 namespace internal {
03517     template<size_t i, size_t m, typename I, typename E = void>
03518     struct bernstein_helper {};
03519
03520     template<typename I>
03521     struct bernstein_helper<0, 0, I> {
03522         using type = typename polynomial<I>::one;
03523     };
03524
03525     template<size_t i, size_t m, typename I>
03526     struct bernstein_helper<i, m, I, std::enable_if_t<
03527         (m > 0) && (i == 0)>> {
03528     private:
03529         using P = polynomial<I>;
03530     public:
03531         using type = typename P::template mul_t<
03532             typename P::template sub_t<typename P::one, typename P::X>,
03533             typename bernstein_helper<i, m-1, I>::type>;
03534     };
03535
03536     template<size_t i, size_t m, typename I>
03537     struct bernstein_helper<i, m, I, std::enable_if_t<
03538         (m > 0) && (i == m)>> {
03539     private:
03540         using P = polynomial<I>;
03541     public:
03542         using type = typename P::template mul_t<
03543             typename P::X,
03544             typename bernstein_helper<i-1, m-1, I>::type>;
03545     };
03546
03547     template<size_t i, size_t m, typename I>
03548     struct bernstein_helper<i, m, I, std::enable_if_t<
03549         (m > 0) && (i > 0) && (i < m)>> {
03550     private:
03551         using P = polynomial<I>;
03552     public:
03553         using type = typename P::template add_t<
03554             typename P::template mul_t<
03555                 typename P::template sub_t<typename P::one, typename P::X>,
03556                 typename bernstein_helper<i, m-1, I>::type>,
03557             typename P::template mul_t<
03558                 typename P::X,

```

```

03559             typename bernstein_helper<i-1, m-1, I>::type>>;
03560         };
03561     } // namespace internal
03562
03563     // AllOne polynomials
03564     namespace internal {
03565         template<size_t deg, typename I>
03566         struct AllOneHelper {
03567             using type = aerobus::add_t<
03568                 typename polynomial<I>::one,
03569                 aerobus::mul_t<
03570                     typename polynomial<I>::X,
03571                     typename AllOneHelper<deg-1, I>::type
03572                 >>;
03573         };
03574
03575         template<typename I>
03576         struct AllOneHelper<0, I> {
03577             using type = typename polynomial<I>::one;
03578         };
03579     } // namespace internal
03580
03581     // Bessel polynomials
03582     namespace internal {
03583         template<size_t deg, typename I>
03584         struct BesselHelper {
03585             private:
03586                 using P = polynomial<I>;
03587                 using factor = typename P::template monomial_t<
03588                     typename I::template inject_constant_t<(2*deg - 1)>,
03589                     1>;
03590             public:
03591                 using type = typename P::template add_t<
03592                     typename P::template mul_t<
03593                         factor,
03594                         typename BesselHelper<deg-1, I>::type
03595                     >,
03596                     typename BesselHelper<deg-2, I>::type
03597                 >;
03598         };
03599
03600         template<typename I>
03601         struct BesselHelper<0, I> {
03602             using type = typename polynomial<I>::one;
03603         };
03604
03605         template<typename I>
03606         struct BesselHelper<1, I> {
03607             private:
03608                 using P = polynomial<I>;
03609             public:
03610                 using type = typename P::template add_t<
03611                     typename P::one,
03612                     typename P::X
03613                 >;
03614         };
03615     } // namespace internal
03616
03617     namespace known_polynomials {
03618         enum hermite_kind {
03619             probabilist,
03620             physicist
03621         };
03622     }
03623
03624     // hermite
03625     namespace internal {
03626         template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03627         struct hermite_helper {};
03628
03629         template<size_t deg, typename I>
03630         struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
03631             private:
03632                 using hnm1 = typename hermite_helper<deg - 1,
03633                     known_polynomials::hermite_kind::probabilist, I>::type;
03634                 using hnm2 = typename hermite_helper<deg - 2,
03635                     known_polynomials::hermite_kind::probabilist, I>::type;
03636             public:
03637                 using type = typename polynomial<I>::template sub_t<
03638                     typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnm1>,
03639                     typename polynomial<I>::template mul_t<
03640                         typename polynomial<I>::template inject_constant_t<deg - 1>,
03641                         hnm2
03642                     >
03643                 >;
03644         };
03645     }
03646

```

```

03647
03648     template<size_t deg, typename I>
03649     struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03650     private:
03651         using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
I>::type;
03652         using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
I>::type;
03653     public:
03654         using type = typename polynomial<I>::template sub_t<
03655             // 2X Hn-1
03656             typename polynomial<I>::template mul_t<
03657                 typename pi64::val<typename I::template inject_constant_t<2>,
03658                 typename I::zero>, hnm1>,
03659                 typename polynomial<I>::template mul_t<
03660                     typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03661                     hnm2
03662                 >
03663             >;
03664     };
03665
03666     template<typename I>
03667     struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03668     private:
03669         using type = typename polynomial<I>::one;
03670     };
03671
03672     template<typename I>
03673     struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03674     private:
03675         using type = typename polynomial<I>::X;
03676     };
03677
03678     template<typename I>
03679     struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03680     private:
03681         using type = typename pi64::one;
03682     };
03683
03684     template<typename I>
03685     struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03686     private:
03687         using type = typename polynomial<I>::template val<
03688             typename I::template inject_constant_t<2>,
03689             typename I::zero>;
03690     };
03691 } // namespace internal
03692
03693 // legendre
03694 namespace internal {
03695     template<size_t n, typename I>
03696     struct legendre_helper {
03697     private:
03698         using Q = FractionField<I>;
03699         using PQ = polynomial<Q>;
03700         // 1/n constant
03701         // (2n-1)/n X
03702         using fact_left = typename PQ::template monomial_t<
03703             typename I::template inject_constant_t<2*n-1>,
03704             typename I::template inject_constant_t<n>
03705         >,
03706         1>;
03707         // (n-1) / n
03708         using fact_right = typename PQ::template val<
03709             typename I::template inject_constant_t<n-1>,
03710             typename I::template inject_constant_t<n>>;
03711     public:
03712         using type = PQ::template sub_t<
03713             typename PQ::template mul_t<
03714                 fact_left,
03715                 typename legendre_helper<n-1, I>::type
03716             >,
03717             typename PQ::template mul_t<
03718                 fact_right,
03719                 typename legendre_helper<n-2, I>::type
03720             >
03721         >;
03722     };
03723
03724     template<typename I>
03725     struct legendre_helper<0, I> {
03726     private:
03727         using type = typename polynomial<FractionField<I>::one>;
03728     };
03729
03730     template<typename I>

```



```

03732     struct legendre_helper<1, I> {
03733         using type = typename polynomial<FractionField<I>::X;
03734     };
03735 } // namespace internal
03736
03737 // bernoulli polynomials
03738 namespace internal {
03739     template<size_t n>
03740     struct bernoulli_coeff {
03741         template<typename T, size_t i>
03742         struct inner {
03743             private:
03744                 using F = FractionField<T>;
03745             public:
03746                 using type = typename F::template mul_t<
03747                     typename F::template inject_ring_t<combination_t<T, i, n>,
03748                     bernoulli_t<T, n-i>
03749                 >;
03750         };
03751     };
03752 } // namespace internal
03753
03754 namespace known_polynomials {
03755     template<size_t deg, typename I = aerobus::i64>
03756     using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
03757
03758     template<size_t deg, typename I = aerobus::i64>
03759     using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
03760
03761     template<size_t deg, typename I = aerobus::i64>
03762     using laguerre = typename internal::laguerre_helper<deg, I>::type;
03763
03764     template<size_t deg, typename I = aerobus::i64>
03765     using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
03766 I>::type;
03767
03768     template<size_t deg, typename I = aerobus::i64>
03769     using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
03770
03771     template<size_t i, size_t m, typename I = aerobus::i64>
03772     using bernstein = typename internal::bernstein_helper<i, m, I>::type;
03773
03774     template<size_t deg, typename I = aerobus::i64>
03775     using legendre = typename internal::legendre_helper<deg, I>::type;
03776
03777     template<size_t deg, typename I = aerobus::i64>
03778     using bernoulli = typename internal::bernoulli_coeff<deg>::template inner, deg>;
03779
03780     template<size_t deg, typename I = aerobus::i64>
03781     using allone = typename internal::AllOneHelper<deg, I>::type;
03782
03783     template<size_t deg, typename I = aerobus::i64>
03784     using bessel = typename internal::BesselHelper<deg, I>::type;
03785 } // namespace known_polynomials
03786 } // namespace aerobus
03787
03788 #ifndef AEROBUS_CONWAY_IMPORTS
03789 // conway polynomials
03790 namespace aerobus {
03791     template<int p, int n>
03792     struct ConwayPolynomial {};
03793
03794 #ifndef DO_NOT_DOCUMENT
03795     #define ZPV ZPV::template val
03796     #define POLYV aerobus::polynomial<ZPV>::template val
03797     template<> struct ConwayPolynomial<2, 1> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<1>; }; // NOLINT
03798     template<> struct ConwayPolynomial<2, 2> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<1>, ZPV<1>; }; // NOLINT
03799     template<> struct ConwayPolynomial<2, 3> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<1>, ZPV<1>; }; // NOLINT
03800     template<> struct ConwayPolynomial<2, 4> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<1>; }; // NOLINT
03801     template<> struct ConwayPolynomial<2, 5> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<1>; }; // NOLINT
03802     template<> struct ConwayPolynomial<2, 6> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<1>, ZPV<1>, ZPV<0>, ZPV<1>, ZPV<1>; }; // NOLINT
03803     template<> struct ConwayPolynomial<2, 7> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<1>; }; // NOLINT
03804     template<> struct ConwayPolynomial<2, 8> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<1>, ZPV<0>, ZPV<0>, ZPV<1>; }; // NOLINT
03805     template<> struct ConwayPolynomial<2, 9> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<0>, ZPV<0>, ZPV<1>; }; //
NOLINT
03806     template<> struct ConwayPolynomial<2, 10> { using ZPV = aerobus::zpv<2>; using type =

```

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

[illegible]

[illegible]

Generated by Doxygen

[illegible]

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

[illegible]

[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

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

```

NOLINT
05793     template<> struct ConwayPolynomial<977, 9> { using ZPZ = aerobus::zpz<977>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<740>, ZPZV<974>;
}; // NOLINT
05794     template<> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<978>; }; // NOLINT
05795     template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5>; }; // NOLINT
05796     template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978>; }; // NOLINT
05797     template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5>; }; // NOLINT
05798     template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<978>; }; // NOLINT
05799     template<> struct ConwayPolynomial<983, 6> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<849>, ZPZV<296>, ZPZV<228>, ZPZV<5>; }; // NOLINT
05800     template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<978>; }; // NOLINT
05801     template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<5>; }; //
NOLINT
05802     template<> struct ConwayPolynomial<983, 9> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<858>, ZPZV<87>, ZPZV<978>;
}; // NOLINT
05803     template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<985>; }; // NOLINT
05804     template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6>; }; // NOLINT
05805     template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985>; }; // NOLINT
05806     template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6>; }; // NOLINT
05807     template<> struct ConwayPolynomial<991, 5> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<985>; }; // NOLINT
05808     template<> struct ConwayPolynomial<991, 6> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<637>, ZPZV<855>, ZPZV<278>, ZPZV<6>; }; // NOLINT
05809     template<> struct ConwayPolynomial<991, 7> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<985>; }; // NOLINT
05810     template<> struct ConwayPolynomial<991, 8> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<941>, ZPZV<786>, ZPZV<234>, ZPZV<6>; }; //
NOLINT
05811     template<> struct ConwayPolynomial<991, 9> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<466>, ZPZV<222>, ZPZV<985>;
}; // NOLINT
05812     template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<990>; }; // NOLINT
05813     template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7>; }; // NOLINT
05814     template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990>; }; // NOLINT
05815     template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7>; }; // NOLINT
05816     template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<990>; }; // NOLINT
05817     template<> struct ConwayPolynomial<997, 6> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<981>, ZPZV<58>, ZPZV<260>, ZPZV<7>; }; // NOLINT
05818     template<> struct ConwayPolynomial<997, 7> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<990>; }; // NOLINT
05819     template<> struct ConwayPolynomial<997, 8> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<934>, ZPZV<473>, ZPZV<241>, ZPZV<7>; }; //
NOLINT
05820     template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<732>, ZPZV<616>, ZPZV<990>;
}; // NOLINT
05821 #endif // DO_NOT_DOCUMENT
05822 } // namespace aerobus
05823 #endif // AEROBUS_CONWAY_IMPORTS
05824
05825 #endif // __INC_AEROBUS__ // NOLINT

```

9.4 src/examples.h File Reference

9.5 examples.h

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

```

00001 #ifndef SRC_EXAMPLES_H_
00002 #define SRC_EXAMPLES_H_
00042 #endif // SRC_EXAMPLES_H_

```


Chapter 10

Examples

10.1 examples/hermite.cpp

How to use `aerobus::known_polynomials::hermite_phys` polynomials

```
#include <cmath>
#include <iostream>
#include "../src/aerobus.h"

namespace standardlib {
    double H3(double x) {
        return 8 * std::pow(x, 3) - 12 * x;
    }

    double H4(double x) {
        return 16 * std::pow(x, 4) - 48 * x * x + 12;
    }
}

namespace aerobuslib {
    double H3(double x) {
        return 8 * aerobus::pow_scalar<double, 3>(x) - 12 * x;
    }

    double H4(double x) {
        return 16 * aerobus::pow_scalar<double, 4>(x) - 48 * x * x + 12;
    }
}

int main() {
    std::cout << std::hermite(3, 10) << '=' << standardlib::H3(10) << '\n'
              << std::hermite(4, 10) << '=' << standardlib::H4(10) << '\n';
    std::cout << aerobus::known_polynomials::hermite_phys<3>::eval(10) << '=' << aerobuslib::H3(10) << '\n'
              << aerobus::known_polynomials::hermite_phys<4>::eval(10) << '=' << aerobuslib::H4(10) << '\n';
}
```

10.2 examples/custom_taylor.cpp

How to implement your own Taylor serie using `aerobus::taylor`

```
#include <cmath>
#include <iostream>
#include <iomanip>
#include "../src/aerobus.h"

template<typename T, size_t i>
struct my_coeff {
    using type = aerobus::makefraction_t<T, aerobus::bell_t<T, i>, aerobus::factorial_t<T, i>>;
};

template<size_t deg>
```



```
#include "../src/aerobus.h"

using FIELD = aerobus::zpz<2>;
using POLYNOMIALS = aerobus::polynomial<FIELD>;
using FRACTIONS = aerobus::FractionField<POLYNOMIALS>;

// x^3 + 2x^2 + 1, with coefficients in Z/2Z, actually x^3 + 1
using P = aerobus::make_int_polynomial_t<FIELD, 1, 2, 0, 1>;
// x^3 + 5x^2 + 7x + 11 with coefficients in Z/17Z, meaning actually x^3 + x^2 + 1
using Q = aerobus::make_int_polynomial_t<FIELD, 1, 5, 8, 1>;

// P/Q in the field of fractions of polynomials
using F = aerobus::makefraction_t<POLYNOMIALS, P, Q>;

int main() {
    const double v = F::eval<double>(1.0);
    std::cout << "expected = " << 2.0/3.0 << std::endl;
    std::cout << "value = " << v << std::endl;
    return 0;
}
```

10.6 examples/make_polynomial.cpp

How to build your own sequence of known polynomials, here [Abel polynomials](#)

```
#include <iostream>
#include "../src/aerobus.h"

// let's build Abel polynomials from scratch using Aerobus

template<typename I = aerobus::i64>
struct AbelHelper {
private:
    using P = aerobus::polynomial<I>;

public:
    // to keep recursion working, we need to operate on a*n and not just a
    template<size_t deg, I::inner_type an>
    struct Inner {
        // abel(n, a) = (x-an) * abel(n-1, a)
        using type = typename aerobus::mul_t<
            typename Inner<deg-1, an>::type,
            typename aerobus::sub_t<typename P::X, typename P::template inject_constant_t<an>>
        >;
    };

    // abel(0, a) = 1
    template<I::inner_type an>
    struct Inner<0, an> {
        using type = P::one;
    };

    // abel(1, a) = X
    template<I::inner_type an>
    struct Inner<1, an> {
        using type = P::X;
    };
};

template<size_t n, auto a, typename I = aerobus::i64>
using AbelPolynomials = typename AbelHelper<I>::template Inner<n, a*n>::type;

using A2_3 = AbelPolynomials<3, 2>;

int main() {
    std::cout << "expected = x^3 - 12 x^2 + 36 x" << std::endl;
    std::cout << "aerobus = " << A2_3::to_string() << std::endl;
    return 0;
}
```

10.7 examples/polynomials_over_finite_field.cpp

How to build a known polynomial (here [aerobus::known_polynomials::allone](#)) with coefficients in a finite field (here [aerobus::zpz<2>](#)) and get its value when evaluated at a value in this field (here 1).

```
#include <iostream>
```

```
#include "../src/aerobus.h"

using GF2 = aerobus::zpz<2>;
using P = aerobus::known_polynomials::allone<8, GF2>;

int main() {
    // at this point, value_at_1 is an instantiation of zpz<2>::val
    using value_at_1 = P::template value_at_t<GF2::template inject_constant_t<1>;
    // here we get its value in an arithmetic type, here int32_t
    constexpr int32_t x = value_at_1::template get<int32_t>();
    // ensure that 1+1+1+1+1+1+1 in  $\mathbb{Z}/2\mathbb{Z}$  is equal to one
    std::cout << "expected = " << 1 << std::endl;
    std::cout << "computed = " << x << std::endl;
    return 0;
}
```

Index

abs_t
 aerobus, [20](#)
add_t
 aerobus, [20](#)
 aerobus::i32, [59](#)
 aerobus::i64, [65](#)
 aerobus::polynomial< Ring >, [73](#)
 aerobus::Quotient< Ring, X >, [81](#)
 aerobus::zpz< p >, [106](#)
addfractions_t
 aerobus, [20](#)
aerobus, [15](#)
 abs_t, [20](#)
 add_t, [20](#)
 addfractions_t, [20](#)
 aligned_malloc, [34](#)
 alternate_t, [20](#)
 alternate_v, [34](#)
 asin, [21](#)
 asinh, [21](#)
 atan, [21](#)
 atanh, [21](#)
 bell_t, [23](#)
 bernoulli_t, [23](#)
 bernoulli_v, [35](#)
 combination_t, [23](#)
 combination_v, [35](#)
 cos, [23](#)
 cosh, [25](#)
 div_t, [25](#)
 E_fraction, [25](#)
 embed_int_poly_in_fractions_t, [25](#)
 exp, [26](#)
 expm1, [26](#)
 factorial_t, [26](#)
 factorial_v, [35](#)
 field, [34](#)
 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, [29](#)
 makefraction_t, [29](#)
 mul_t, [29](#)
 mulfractions_t, [29](#)
 pi64, [30](#)
 PI_fraction, [30](#)
 pow_t, [30](#)
 pq64, [30](#)
 q32, [30](#)
 q64, [31](#)
 sin, [31](#)
 sinh, [31](#)
 SQRT2_fraction, [31](#)
 SQRT3_fraction, [31](#)
 stirling_signed_t, [32](#)
 stirling_unsigned_t, [32](#)
 sub_t, [32](#)
 tan, [32](#)
 tanh, [33](#)
 taylor, [33](#)
 vadd_t, [33](#)
 vmul_t, [34](#)
aerobus::ContinuedFraction< a0 >, [49](#)
 type, [49](#)
 val, [50](#)
aerobus::ContinuedFraction< a0, rest... >, [50](#)
 type, [51](#)
 val, [51](#)
aerobus::ContinuedFraction< values >, [48](#)
aerobus::ConwayPolynomial, [51](#)
aerobus::Embed< i32, i64 >, [52](#)
 type, [52](#)
aerobus::Embed< polynomial< Small >, polynomial< Large > >, [53](#)
 type, [53](#)
aerobus::Embed< q32, q64 >, [54](#)
 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, [58](#)
 add_t, [59](#)
 div_t, [59](#)
 eq_t, [59](#)
 eq_v, [62](#)
 gcd_t, [60](#)
 gt_t, [60](#)
 inject_constant_t, [60](#)

- inject_ring_t, 60
- inner_type, 60
- is_euclidean_domain, 62
- is_field, 63
- lt_t, 61
- mod_t, 61
- mul_t, 61
- one, 61
- pos_t, 61
- pos_v, 63
- sub_t, 62
- zero, 62
- aerobus::i32::val< x >, 90
 - enclosing_type, 91
 - get, 91
 - is_zero_t, 91
 - to_string, 91
 - v, 91
- aerobus::i64, 63
 - add_t, 65
 - div_t, 65
 - eq_t, 65
 - eq_v, 68
 - gcd_t, 65
 - gt_t, 66
 - gt_v, 68
 - inject_constant_t, 66
 - inject_ring_t, 66
 - inner_type, 66
 - is_euclidean_domain, 69
 - is_field, 69
 - lt_t, 66
 - lt_v, 69
 - mod_t, 67
 - mul_t, 67
 - one, 67
 - pos_t, 67
 - pos_v, 69
 - sub_t, 68
 - zero, 68
- aerobus::i64::val< x >, 92
 - enclosing_type, 93
 - get, 93
 - inner_type, 93
 - is_zero_t, 93
 - to_string, 93
 - v, 94
- aerobus::internal, 36
 - index_sequence_reverse, 39
 - is_instantiation_of_v, 40
 - make_index_sequence_reverse, 39
 - type_at_t, 39
- aerobus::is_prime< n >, 71
 - value, 71
- aerobus::IsEuclideanDomain, 45
- aerobus::IsField, 45
- aerobus::IsRing, 46
- aerobus::known_polynomials, 40
- allone, 41
- bernoulli, 41
- bernstein, 41
- chebyshev_T, 41
- chebyshev_U, 42
- hermite_kind, 44
- hermite_phys, 42
- hermite_prob, 43
- laguerre, 43
- legendre, 43
- physicist, 44
- probabilist, 44
- aerobus::polynomial< Ring >, 71
 - add_t, 73
 - derive_t, 73
 - div_t, 73
 - eq_t, 74
 - gcd_t, 74
 - gt_t, 74
 - inject_constant_t, 75
 - inject_ring_t, 75
 - is_euclidean_domain, 79
 - is_field, 79
 - lt_t, 75
 - mod_t, 75
 - monomial_t, 76
 - mul_t, 76
 - one, 76
 - pos_t, 76
 - pos_v, 79
 - simplify_t, 78
 - sub_t, 78
 - X, 78
 - zero, 78
- aerobus::polynomial< Ring >::horner_reduction_t< P >, 57
- aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop >, 69
 - type, 70
- aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< stop, stop >, 70
 - type, 70
- aerobus::polynomial< Ring >::val< coeffN >, 101
 - aN, 103
 - coeff_at_t, 103
 - degree, 104
 - enclosing_type, 103
 - eval, 104
 - is_zero_t, 103
 - is_zero_v, 104
 - ring_type, 103
 - strip, 103
 - to_string, 104
 - value_at_t, 103
- 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
- type, 47
- aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> >, 48
- type, 48
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 94
- aN, 95
- coeff_at_t, 95
- degree, 97
- enclosing_type, 95
- eval, 96
- is_zero_t, 96
- is_zero_v, 97
- ring_type, 96
- strip, 96
- to_string, 97
- value_at_t, 96
- aerobus::Quotient< Ring, X >, 80
- add_t, 81
- div_t, 82
- eq_t, 82
- eq_v, 84
- inject_constant_t, 82
- inject_ring_t, 82
- is_euclidean_domain, 84
- mod_t, 83
- mul_t, 83
- one, 83
- pos_t, 83
- pos_v, 84
- zero, 84
- aerobus::Quotient< Ring, X >::val< V >, 98
- raw_t, 98
- type, 98
- aerobus::type_list< Ts >, 86
- at, 87
- concat, 87
- insert, 87
- length, 88
- push_back, 87
- push_front, 88
- remove, 88
- aerobus::type_list< Ts >::pop_front, 79
- tail, 80
- type, 80
- aerobus::type_list< Ts >::split< index >, 85
- head, 85
- tail, 85
- aerobus::type_list<>, 89
- concat, 89
- insert, 89
- length, 90
- push_back, 89
- push_front, 89
- aerobus::zpz< p >, 105
- add_t, 106
- div_t, 107
- eq_t, 107
- eq_v, 110
- gcd_t, 107
- gt_t, 107
- gt_v, 110
- inject_constant_t, 108
- inner_type, 108
- is_euclidean_domain, 111
- is_field, 111
- lt_t, 108
- lt_v, 111
- mod_t, 108
- mul_t, 109
- one, 109
- pos_t, 109
- pos_v, 111
- sub_t, 109
- zero, 110
- aerobus::zpz< p >::val< x >, 98
- enclosing_type, 99
- get, 100
- is_zero_t, 99
- is_zero_v, 100
- to_string, 100
- v, 100
- aligned_malloc
- aerobus, 34
- allone
- aerobus::known_polynomials, 41
- alternate_t
- aerobus, 20
- alternate_v
- aerobus, 34
- aN
- aerobus::polynomial< Ring >::val< coeffN >, 103
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 95
- asin
- aerobus, 21
- asinh
- aerobus, 21
- at
- aerobus::type_list< Ts >, 87
- atan
- aerobus, 21
- atanh
- aerobus, 21
- bell_t
- aerobus, 23
- bernoulli
- aerobus::known_polynomials, 41
- bernoulli_t
- aerobus, 23
- bernoulli_v
- aerobus, 35
- bernstein
- aerobus::known_polynomials, 41

- chebyshev_T
 - aerobus::known_polynomials, 41
- chebyshev_U
 - aerobus::known_polynomials, 42
- coeff_at_t
 - aerobus::polynomial< Ring >::val< coeffN >, 103
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 95
- combination_t
 - aerobus, 23
- combination_v
 - aerobus, 35
- concat
 - aerobus::type_list< Ts >, 87
 - aerobus::type_list<>, 89
- cos
 - aerobus, 23
- cosh
 - aerobus, 25
- degree
 - aerobus::polynomial< Ring >::val< coeffN >, 104
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 97
- derive_t
 - aerobus::polynomial< Ring >, 73
- div_t
 - aerobus, 25
 - aerobus::i32, 59
 - aerobus::i64, 65
 - aerobus::polynomial< Ring >, 73
 - aerobus::Quotient< Ring, X >, 82
 - aerobus::zpz< p >, 107
- E_fraction
 - aerobus, 25
- embed_int_poly_in_fractions_t
 - aerobus, 25
- enclosing_type
 - aerobus::i32::val< x >, 91
 - aerobus::i64::val< x >, 93
 - aerobus::polynomial< Ring >::val< coeffN >, 103
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 95
 - aerobus::zpz< p >::val< x >, 99
- eq_t
 - aerobus::i32, 59
 - aerobus::i64, 65
 - aerobus::polynomial< Ring >, 74
 - aerobus::Quotient< Ring, X >, 82
 - aerobus::zpz< p >, 107
- eq_v
 - aerobus::i32, 62
 - aerobus::i64, 68
 - aerobus::Quotient< Ring, X >, 84
 - aerobus::zpz< p >, 110
- eval
 - aerobus::polynomial< Ring >::val< coeffN >, 104
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 96
- exp
 - aerobus, 26
- expm1
 - aerobus, 26
- factorial_t
 - aerobus, 26
- factorial_v
 - aerobus, 35
- field
 - aerobus, 34
- fpq32
 - aerobus, 26
- fpq64
 - aerobus, 27
- FractionField
 - aerobus, 27
- gcd_t
 - aerobus, 27
 - aerobus::i32, 60
 - aerobus::i64, 65
 - aerobus::polynomial< Ring >, 74
 - aerobus::zpz< p >, 107
- geometric_sum
 - aerobus, 27
- get
 - aerobus::i32::val< x >, 91
 - aerobus::i64::val< x >, 93
 - aerobus::zpz< p >::val< x >, 100
- gt_t
 - aerobus::i32, 60
 - aerobus::i64, 66
 - aerobus::polynomial< Ring >, 74
 - aerobus::zpz< p >, 107
- gt_v
 - aerobus::i64, 68
 - aerobus::zpz< p >, 110
- head
 - aerobus::type_list< Ts >::split< index >, 85
- hermite_kind
 - aerobus::known_polynomials, 44
- hermite_phys
 - aerobus::known_polynomials, 42
- hermite_prob
 - aerobus::known_polynomials, 43
- index_sequence_reverse
 - aerobus::internal, 39
- inject_constant_t
 - aerobus::i32, 60
 - aerobus::i64, 66
 - aerobus::polynomial< Ring >, 75
 - aerobus::Quotient< Ring, X >, 82
 - aerobus::zpz< p >, 108
- inject_ring_t

- aerobus::i32, 60
- aerobus::i64, 66
- aerobus::polynomial< Ring >, 75
- aerobus::Quotient< Ring, X >, 82
- inner_type
 - aerobus::i32, 60
 - aerobus::i64, 66
 - aerobus::i64::val< x >, 93
 - aerobus::zpz< p >, 108
- insert
 - aerobus::type_list< Ts >, 87
 - aerobus::type_list<>, 89
- Introduction, 1
- is_euclidean_domain
 - aerobus::i32, 62
 - aerobus::i64, 69
 - aerobus::polynomial< Ring >, 79
 - aerobus::Quotient< Ring, X >, 84
 - aerobus::zpz< p >, 111
- is_field
 - aerobus::i32, 63
 - aerobus::i64, 69
 - aerobus::polynomial< Ring >, 79
 - aerobus::zpz< p >, 111
- is_instantiation_of_v
 - aerobus::internal, 40
- is_zero_t
 - aerobus::i32::val< x >, 91
 - aerobus::i64::val< x >, 93
 - aerobus::polynomial< Ring >::val< coeffN >, 103
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 96
 - aerobus::zpz< p >::val< x >, 99
- is_zero_v
 - aerobus::polynomial< Ring >::val< coeffN >, 104
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 97
 - aerobus::zpz< p >::val< x >, 100
- laguerre
 - aerobus::known_polynomials, 43
- legendre
 - aerobus::known_polynomials, 43
- length
 - aerobus::type_list< Ts >, 88
 - aerobus::type_list<>, 90
- Inp1
 - aerobus, 27
- lt_t
 - aerobus::i32, 61
 - aerobus::i64, 66
 - aerobus::polynomial< Ring >, 75
 - aerobus::zpz< p >, 108
- lt_v
 - aerobus::i64, 69
 - aerobus::zpz< p >, 111
- make_frac_polynomial_t
 - aerobus, 28
- make_index_sequence_reverse
 - aerobus::internal, 39
- make_int_polynomial_t
 - aerobus, 28
- make_q32_t
 - aerobus, 28
- make_q64_t
 - aerobus, 29
- makefraction_t
 - aerobus, 29
- mod_t
 - aerobus::i32, 61
 - aerobus::i64, 67
 - aerobus::polynomial< Ring >, 75
 - aerobus::Quotient< Ring, X >, 83
 - aerobus::zpz< p >, 108
- monomial_t
 - aerobus::polynomial< Ring >, 76
- mul_t
 - aerobus, 29
 - aerobus::i32, 61
 - aerobus::i64, 67
 - aerobus::polynomial< Ring >, 76
 - aerobus::Quotient< Ring, X >, 83
 - aerobus::zpz< p >, 109
- mulfractions_t
 - aerobus, 29
- one
 - aerobus::i32, 61
 - aerobus::i64, 67
 - aerobus::polynomial< Ring >, 76
 - aerobus::Quotient< Ring, X >, 83
 - aerobus::zpz< p >, 109
- physicist
 - aerobus::known_polynomials, 44
- pi64
 - aerobus, 30
- PI_fraction
 - aerobus, 30
- pos_t
 - aerobus::i32, 61
 - aerobus::i64, 67
 - aerobus::polynomial< Ring >, 76
 - aerobus::Quotient< Ring, X >, 83
 - aerobus::zpz< p >, 109
- pos_v
 - aerobus::i32, 63
 - aerobus::i64, 69
 - aerobus::polynomial< Ring >, 79
 - aerobus::Quotient< Ring, X >, 84
 - aerobus::zpz< p >, 111
- pow_t
 - aerobus, 30
- pq64
 - aerobus, 30
- probabilist
 - aerobus::known_polynomials, 44

- push_back
 - aerobus::type_list< Ts >, 87
 - aerobus::type_list<>, 89
- push_front
 - aerobus::type_list< Ts >, 88
 - aerobus::type_list<>, 89
- q32
 - aerobus, 30
- q64
 - aerobus, 31
- raw_t
 - aerobus::Quotient< Ring, X >::val< V >, 98
- README.md, 113
- remove
 - aerobus::type_list< Ts >, 88
- ring_type
 - aerobus::polynomial< Ring >::val< coeffN >, 103
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 96
- simplify_t
 - aerobus::polynomial< Ring >, 78
- sin
 - aerobus, 31
- sinh
 - aerobus, 31
- SQRT2_fraction
 - aerobus, 31
- SQRT3_fraction
 - aerobus, 31
- src/aerobus.h, 113
- src/examples.h, 203
- stirling_signed_t
 - aerobus, 32
- stirling_unsigned_t
 - aerobus, 32
- strip
 - aerobus::polynomial< Ring >::val< coeffN >, 103
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 96
- sub_t
 - aerobus, 32
 - aerobus::i32, 62
 - aerobus::i64, 68
 - aerobus::polynomial< Ring >, 78
 - aerobus::zpz< p >, 109
- tail
 - aerobus::type_list< Ts >::pop_front, 80
 - aerobus::type_list< Ts >::split< index >, 85
- tan
 - aerobus, 32
- tanh
 - aerobus, 33
- taylor
 - aerobus, 33
- to_string
 - aerobus::i32::val< x >, 91
 - aerobus::i64::val< x >, 93
 - aerobus::polynomial< Ring >::val< coeffN >, 104
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 97
 - aerobus::zpz< p >::val< x >, 100
- type
 - aerobus::ContinuedFraction< a0 >, 49
 - aerobus::ContinuedFraction< a0, rest... >, 51
 - 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 >::horner_reduction_t< P >::inner< index, stop >, 70
 - aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< stop, stop >, 70
 - 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 >, 98
 - aerobus::type_list< Ts >::pop_front, 80
- type_at_t
 - aerobus::internal, 39
- v
 - aerobus::i32::val< x >, 91
 - aerobus::i64::val< x >, 94
 - aerobus::zpz< p >::val< x >, 100
- vadd_t
 - aerobus, 33
- val
 - aerobus::ContinuedFraction< a0 >, 50
 - aerobus::ContinuedFraction< a0, rest... >, 51
- value
 - aerobus::is_prime< n >, 71
- value_at_t
 - aerobus::polynomial< Ring >::val< coeffN >, 103
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 96
- vmul_t
 - aerobus, 34
- X
 - aerobus::polynomial< Ring >, 78
- zero
 - aerobus::i32, 62
 - aerobus::i64, 68
 - aerobus::polynomial< Ring >, 78
 - aerobus::Quotient< Ring, X >, 84

aerobus::zpz< p >, [110](#)