

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_1_signed_t	32
6.1.2.44 stirling_1_unsigned_t	32
6.1.2.45 stirling_2_t	32
6.1.2.46 sub_t	33
6.1.2.47 tan	33
6.1.2.48 tanh	33
6.1.2.49 taylor	33
6.1.2.50 vadd_t	34
6.1.2.51 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	35
6.1.4.1 alternate_v	35
6.1.4.2 bernoulli_v	35
6.1.4.3 combination_v	35
6.1.4.4 factorial_v	36
6.2 aerobus::internal Namespace Reference	36
6.2.1 Detailed Description	39
6.2.2 Typedef Documentation	40
6.2.2.1 make_index_sequence_reverse	40
6.2.2.2 type_at_t	40
6.2.3 Function Documentation	40
6.2.3.1 index_sequence_reverse()	40
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 Enumeration Type Documentation	40
6.3.2.1 hermite_kind	40
7 Concept Documentation	43
7.1 aerobus::IsEuclideanDomain Concept Reference	43
7.1.1 Concept definition	43
7.1.2 Detailed Description	43
7.2 aerobus::IsField Concept Reference	43
7.2.1 Concept definition	43
7.2.2 Detailed Description	44
7.3 aerobus::IsRing Concept Reference	44
7.3.1 Concept definition	44
7.3.2 Detailed Description	44
8 Class Documentation	45
8.1 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E > Struct Template Reference	45
8.2 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0 index > 0)> > Struct Template Reference	45
8.2.1 Member Typedef Documentation	45
8.2.1.1 type	45
8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference	46
8.3.1 Member Typedef Documentation	46
8.3.1.1 type	46
8.4 aerobus::ContinuedFraction< values > Struct Template Reference	46
8.4.1 Detailed Description	46

8.5 aerobus::ContinuedFraction< a0 > Struct Template Reference	47
8.5.1 Detailed Description	47
8.5.2 Member Typedef Documentation	47
8.5.2.1 type	47
8.5.3 Member Data Documentation	48
8.5.3.1 val	48
8.6 aerobus::ContinuedFraction< a0, rest... > Struct Template Reference	48
8.6.1 Detailed Description	48
8.6.2 Member Typedef Documentation	49
8.6.2.1 type	49
8.6.3 Member Data Documentation	49
8.6.3.1 val	49
8.7 aerobus::ConwayPolynomial Struct Reference	49
8.8 aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner< index, ghost > Struct Template Reference	49
8.8.1 Member Function Documentation	50
8.8.1.1 func()	50
8.9 aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner<-1, ghost > Struct Template Reference	50
8.9.1 Member Function Documentation	50
8.9.1.1 func()	50
8.10 aerobus::Embed< Small, Large, E > Struct Template Reference	51
8.10.1 Detailed Description	51
8.11 aerobus::Embed< i32, i64 > Struct Reference	51
8.11.1 Detailed Description	51
8.11.2 Member Typedef Documentation	51
8.11.2.1 type	51
8.12 aerobus::Embed< polynomial< Small >, polynomial< Large > > Struct Template Reference	52
8.12.1 Detailed Description	52
8.12.2 Member Typedef Documentation	52
8.12.2.1 type	52
8.13 aerobus::Embed< q32, q64 > Struct Reference	53
8.13.1 Detailed Description	53
8.13.2 Member Typedef Documentation	53
8.13.2.1 type	53
8.14 aerobus::Embed< Quotient< Ring, X >, Ring > Struct Template Reference	54
8.14.1 Detailed Description	54
8.14.2 Member Typedef Documentation	54
8.14.2.1 type	54
8.15 aerobus::Embed< Ring, FractionField< Ring > > Struct Template Reference	55
8.15.1 Detailed Description	55
8.15.2 Member Typedef Documentation	55
8.15.2.1 type	55

8.16 aerobus::Embed< zpz< x >, i32 > Struct Template Reference	55
8.16.1 Detailed Description	56
8.16.2 Member Typedef Documentation	56
8.16.2.1 type	56
8.17 aerobus::polynomial< Ring >::horner_reduction_t< P > Struct Template Reference	56
8.17.1 Detailed Description	57
8.18 aerobus::i32 Struct Reference	57
8.18.1 Detailed Description	58
8.18.2 Member Typedef Documentation	59
8.18.2.1 add_t	59
8.18.2.2 div_t	59
8.18.2.3 eq_t	59
8.18.2.4 gcd_t	59
8.18.2.5 gt_t	60
8.18.2.6 inject_constant_t	60
8.18.2.7 inject_ring_t	60
8.18.2.8 inner_type	60
8.18.2.9 lt_t	60
8.18.2.10 mod_t	61
8.18.2.11 mul_t	61
8.18.2.12 one	61
8.18.2.13 pos_t	61
8.18.2.14 sub_t	62
8.18.2.15 zero	62
8.18.3 Member Data Documentation	62
8.18.3.1 eq_v	62
8.18.3.2 is_euclidean_domain	62
8.18.3.3 is_field	62
8.18.3.4 pos_v	63
8.19 aerobus::i64 Struct Reference	64
8.19.1 Detailed Description	65
8.19.2 Member Typedef Documentation	65
8.19.2.1 add_t	65
8.19.2.2 div_t	66
8.19.2.3 eq_t	66
8.19.2.4 gcd_t	66
8.19.2.5 gt_t	66
8.19.2.6 inject_constant_t	67
8.19.2.7 inject_ring_t	67
8.19.2.8 inner_type	67
8.19.2.9 lt_t	67
8.19.2.10 mod_t	68

8.19.2.11 mul_t	68
8.19.2.12 one	68
8.19.2.13 pos_t	68
8.19.2.14 sub_t	68
8.19.2.15 zero	69
8.19.3 Member Data Documentation	69
8.19.3.1 eq_v	69
8.19.3.2 gt_v	69
8.19.3.3 is_euclidean_domain	69
8.19.3.4 is_field	70
8.19.3.5 lt_v	70
8.19.3.6 pos_v	70
8.20 aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop > Struct Template Reference	70
8.20.1 Member Typedef Documentation	71
8.20.1.1 type	71
8.21 aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< stop, stop > Struct Template Reference	71
8.21.1 Member Typedef Documentation	71
8.21.1.1 type	71
8.22 aerobus::is_prime< n > Struct Template Reference	71
8.22.1 Detailed Description	72
8.22.2 Member Data Documentation	72
8.22.2.1 value	72
8.23 aerobus::polynomial< Ring > Struct Template Reference	72
8.23.1 Detailed Description	74
8.23.2 Member Typedef Documentation	74
8.23.2.1 add_t	74
8.23.2.2 derive_t	74
8.23.2.3 div_t	75
8.23.2.4 eq_t	75
8.23.2.5 gcd_t	75
8.23.2.6 gt_t	75
8.23.2.7 inject_constant_t	76
8.23.2.8 inject_ring_t	76
8.23.2.9 lt_t	76
8.23.2.10 mod_t	76
8.23.2.11 monomial_t	77
8.23.2.12 mul_t	77
8.23.2.13 one	77
8.23.2.14 pos_t	77
8.23.2.15 simplify_t	79
8.23.2.16 sub_t	79

8.23.2.17 X	79
8.23.2.18 zero	79
8.23.3 Member Data Documentation	80
8.23.3.1 is_euclidean_domain	80
8.23.3.2 is_field	80
8.23.3.3 pos_v	80
8.24 aerobus::type_list< Ts >::pop_front Struct Reference	80
8.24.1 Detailed Description	80
8.24.2 Member Typedef Documentation	81
8.24.2.1 tail	81
8.24.2.2 type	81
8.25 aerobus::Quotient< Ring, X > Struct Template Reference	81
8.25.1 Detailed Description	82
8.25.2 Member Typedef Documentation	82
8.25.2.1 add_t	82
8.25.2.2 div_t	83
8.25.2.3 eq_t	83
8.25.2.4 inject_constant_t	83
8.25.2.5 inject_ring_t	84
8.25.2.6 mod_t	84
8.25.2.7 mul_t	84
8.25.2.8 one	84
8.25.2.9 pos_t	85
8.25.2.10 zero	85
8.25.3 Member Data Documentation	85
8.25.3.1 eq_v	85
8.25.3.2 is_euclidean_domain	85
8.25.3.3 pos_v	85
8.26 aerobus::type_list< Ts >::split< index > Struct Template Reference	86
8.26.1 Detailed Description	86
8.26.2 Member Typedef Documentation	86
8.26.2.1 head	86
8.26.2.2 tail	86
8.27 aerobus::type_list< Ts > Struct Template Reference	87
8.27.1 Detailed Description	87
8.27.2 Member Typedef Documentation	88
8.27.2.1 at	88
8.27.2.2 concat	88
8.27.2.3 insert	88
8.27.2.4 push_back	89
8.27.2.5 push_front	89
8.27.2.6 remove	89

8.27.3 Member Data Documentation	89
8.27.3.1 length	89
8.28 aerobus::type_list<> Struct Reference	90
8.28.1 Detailed Description	90
8.28.2 Member Typedef Documentation	90
8.28.2.1 concat	90
8.28.2.2 insert	90
8.28.2.3 push_back	90
8.28.2.4 push_front	90
8.28.3 Member Data Documentation	91
8.28.3.1 length	91
8.29 aerobus::i32::val< x > Struct Template Reference	91
8.29.1 Detailed Description	91
8.29.2 Member Typedef Documentation	92
8.29.2.1 enclosing_type	92
8.29.2.2 is_zero_t	92
8.29.3 Member Function Documentation	92
8.29.3.1 get()	92
8.29.3.2 to_string()	92
8.29.4 Member Data Documentation	92
8.29.4.1 v	92
8.30 aerobus::i64::val< x > Struct Template Reference	93
8.30.1 Detailed Description	93
8.30.2 Member Typedef Documentation	94
8.30.2.1 enclosing_type	94
8.30.2.2 inner_type	94
8.30.2.3 is_zero_t	94
8.30.3 Member Function Documentation	94
8.30.3.1 get()	94
8.30.3.2 to_string()	94
8.30.4 Member Data Documentation	95
8.30.4.1 v	95
8.31 aerobus::polynomial< Ring >::val< coeffN, coeffs > Struct Template Reference	95
8.31.1 Detailed Description	96
8.31.2 Member Typedef Documentation	96
8.31.2.1 aN	96
8.31.2.2 coeff_at_t	96
8.31.2.3 enclosing_type	97
8.31.2.4 is_zero_t	97
8.31.2.5 ring_type	97
8.31.2.6 strip	97
8.31.2.7 value_at_t	97

8.31.3 Member Function Documentation	97
8.31.3.1 compensated_eval()	97
8.31.3.2 eval()	98
8.31.3.3 to_string()	98
8.31.4 Member Data Documentation	99
8.31.4.1 degree	99
8.31.4.2 is_zero_v	99
8.32 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference	99
8.32.1 Detailed Description	99
8.32.2 Member Typedef Documentation	100
8.32.2.1 raw_t	100
8.32.2.2 type	100
8.33 aerobus::zpz< p >::val< x > Struct Template Reference	100
8.33.1 Detailed Description	100
8.33.2 Member Typedef Documentation	101
8.33.2.1 enclosing_type	101
8.33.2.2 is_zero_t	101
8.33.3 Member Function Documentation	101
8.33.3.1 get()	101
8.33.3.2 to_string()	101
8.33.4 Member Data Documentation	102
8.33.4.1 is_zero_v	102
8.33.4.2 v	102
8.34 aerobus::polynomial< Ring >::val< coeffN > Struct Template Reference	102
8.34.1 Detailed Description	103
8.34.2 Member Typedef Documentation	103
8.34.2.1 aN	103
8.34.2.2 coeff_at_t	103
8.34.2.3 enclosing_type	103
8.34.2.4 is_zero_t	104
8.34.2.5 ring_type	104
8.34.2.6 strip	104
8.34.2.7 value_at_t	104
8.34.3 Member Function Documentation	104
8.34.3.1 compensated_eval()	104
8.34.3.2 eval()	104
8.34.3.3 to_string()	105
8.34.4 Member Data Documentation	105
8.34.4.1 degree	105
8.34.4.2 is_zero_v	105
8.35 aerobus::zpz< p > Struct Template Reference	105
8.35.1 Detailed Description	107

8.35.2 Member Typedef Documentation	107
8.35.2.1 add_t	107
8.35.2.2 div_t	107
8.35.2.3 eq_t	108
8.35.2.4 gcd_t	108
8.35.2.5 gt_t	108
8.35.2.6 inject_constant_t	109
8.35.2.7 inner_type	109
8.35.2.8 lt_t	109
8.35.2.9 mod_t	109
8.35.2.10 mul_t	110
8.35.2.11 one	110
8.35.2.12 pos_t	110
8.35.2.13 sub_t	110
8.35.2.14 zero	111
8.35.3 Member Data Documentation	111
8.35.3.1 eq_v	111
8.35.3.2 gt_v	111
8.35.3.3 is_euclidean_domain	111
8.35.3.4 is_field	111
8.35.3.5 lt_v	112
8.35.3.6 pos_v	112
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	207
9.5 examples.h	207
10 Examples	209
10.1 examples/hermite.cpp	209
10.2 examples/custom_taylor.cpp	209
10.3 examples/fp16.cu	210
10.4 examples/continued_fractions.cpp	213
10.5 examples/modular_arithmetic.cpp	213
10.6 examples/make_polynomial.cpp	213
10.7 examples/polynomials_over_finite_field.cpp	214
10.8 examples/compensated_horner.cpp	214
Index	217

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](#). And Google's [Benchmark library](#). Then move to top directory :

```
rm -rf build
mkdir build
cd build
cmake ..
make benchmarks
./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::zpz<p>` : integers modulo p (prime number) on 32 bits

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

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

and the following "elements" :

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

1.2.2 Polynomials

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

In addition, values have an evaluation function :

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

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

1.2.3 Known polynomials

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

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

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

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

1.2.4 Conway polynomials

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

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

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

1.2.5 Taylor series

Aerobus provides definition for Taylor expansion of known functions. They are all templates in two parameters, degree of expansion (`size_t`) and Integers (`typename`). Coefficients then live in `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 = zp<8>;
using Fzmz = FractionField<ZmZ>;
using Pfzmz = polynomial<Fzmz>;
```

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

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

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

Which also have an evaluation function, as polynomial do.

1.3.2 Quotient

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

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

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

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

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

1.4 Misc

1.4.1 Continued Fractions

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

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

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

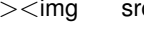
```
constexpr double A_SQRT3 = aerobus::SQRT3_fraction::val; // 1.7320508075688772935
```

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 easy to make it work for `__half2` because of [another bug](#).

One workaround is to add `constexpr` modifier on line 5039 of file `cuda_fp16.h`. Once done, `examples\fp16.cu` compiles and generates proper assembly. Please push to make these bug fixed by NVIDIA. <https://zenodo.org/badge/latestdoi/499577459>  <https://zenodo.org/badge/499577459.svg> alt="DOI"/>

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	43
aerobus::IsField	
Concept to express R is a field	43
aerobus::IsRing	
Concept to express R is a Ring	44

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 >	45
aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0 index > 0)> > 45	
aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> >	46
aerobus::ContinuedFraction< values >	
Continued fraction $a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots}}$	46
aerobus::ContinuedFraction< a0 >	
Specialization for only one coefficient, technically just 'a0'	47
aerobus::ContinuedFraction< a0, rest... >	
Specialization for multiple coefficients (strictly more than one)	48
aerobus::ConwayPolynomial	49
aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner< index, ghost >	49
aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner<-1, ghost > . .	50
aerobus::Embed< Small, Large, E >	
Embedding - struct forward declaration	51
aerobus::Embed< i32, i64 >	
Embeds i32 into i64	51
aerobus::Embed< polynomial< Small >, polynomial< Large > >	
Embeds polynomial<Small> into polynomial<Large>	52
aerobus::Embed< q32, q64 >	
Embeds q32 into q64	53
aerobus::Embed< Quotient< Ring, X >, Ring >	
Embeds Quotient<Ring, X> into Ring	54
aerobus::Embed< Ring, FractionField< Ring > >	
Embeds values from Ring to its field of fractions	55
aerobus::Embed< zpz< x >, i32 >	
Embeds zpz values into i32	55
aerobus::polynomial< Ring >::horner_reduction_t< P >	
Used to evaluate polynomials over a value in Ring	56
aerobus::i32	
32 bits signed integers, seen as a algebraic ring with related operations	57
aerobus::i64	
64 bits signed integers, seen as a algebraic ring with related operations	64
aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop >	70
aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< stop, stop >	71

aerobus::is_prime< n >	71
Checks if n is prime	
aerobus::polynomial< Ring >	72
aerobus::type_list< Ts >::pop_front	
Removes types from head of the list	80
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 $\mathbb{Z}/2\mathbb{Z}$	81
aerobus::type_list< Ts >::split< index >	
Splits list at index	86
aerobus::type_list< Ts >	
Empty pure template struct to handle type list	87
aerobus::type_list<>	
Specialization for empty type list	90
aerobus::i32::val< x >	
Values in i32 , again represented as types	91
aerobus::i64::val< x >	
Values in i64	93
aerobus::polynomial< Ring >::val< coeffN, coeffs >	
Values (seen as types) in polynomial ring	95
aerobus::Quotient< Ring, X >::val< V >	
Projection values in the quotient ring	99
aerobus::zpz< p >::val< x >	
Values in zpz	100
aerobus::polynomial< Ring >::val< coeffN >	
Specialization for constants	102
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	207

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 (v1 + v2 + ... + vn) 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 (v1 * v2 + ... * vn) 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 Q for 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_1_signed_t = typename internal::stirling_1_helper< T, n, k >::type`
Stirling number of first kind (signed) – as types.
- `template<typename T, int n, int k>`
`using stirling_1_unsigned_t = abs_t< typename internal::stirling_1_helper< T, n, k >::type >`
Stirling number of first kind (unsigned) – as types.
- `template<typename T, int n, int k>`
`using stirling_2_t = typename internal::stirling_2_helper< T, n, k >::type`
Stirling number of second kind – as types.
- `template<typename T, typename p, size_t n>`
`using pow_t = typename internal::pow< T, p, n >::type`
 p^n (as 'val' type in T)
- `template<typename T, template< typename, size_t index > typename coeff_at, size_t deg>`
`using taylor = typename internal::make_taylor_impl< T, coeff_at, internal::make_index_sequence_reverse< deg+1 > >::type`
- `template<typename Integers, size_t deg>`
`using exp = taylor< Integers, internal::exp_coeff, deg >`
 e^x
- `template<typename Integers, size_t deg>`
`using expm1 = typename polynomial< FractionField< Integers > >::template sub_t< exp< Integers, deg >, typename polynomial< FractionField< Integers > >::one >`
 $e^x - 1$
- `template<typename Integers, size_t deg>`
`using lnp1 = taylor< Integers, internal::lnp1_coeff, deg >`
 $\ln(1+x)$
- `template<typename Integers, size_t deg>`
`using atan = taylor< Integers, internal::atan_coeff, deg >`
 $\arctan(x)$
- `template<typename Integers, size_t deg>`
`using sin = taylor< Integers, internal::sin_coeff, deg >`
 $\sin(x)$
- `template<typename Integers, size_t deg>`
`using sinh = taylor< Integers, internal::sh_coeff, deg >`
 $\sinh(x)$
- `template<typename Integers, size_t deg>`
`using cosh = taylor< Integers, internal::cosh_coeff, deg >`
 $\cosh(x)$ *hyperbolic cosine*
- `template<typename Integers, size_t deg>`
`using cos = taylor< Integers, internal::cos_coeff, deg >`
 $\cos(x)$ *cosinus*
- `template<typename Integers, size_t deg>`
`using geometric_sum = taylor< Integers, internal::geom_coeff, deg >`

- $\frac{1}{1-x}$ zero development of $\frac{1}{1-x}$
 • template<typename Integers , size_t deg>
 using **asin** = **taylor**< Integers, internal::asin_coeff, deg >
 arcsin(x) *arc sinus*
- template<typename Integers , size_t deg>
 using **asinh** = **taylor**< Integers, internal::asinh_coeff, deg >
 arcsinh(x) *arc hyperbolic sinus*
- template<typename Integers , size_t deg>
 using **atanh** = **taylor**< Integers, internal::atanh_coeff, deg >
 arctanh(x) *arc hyperbolic tangent*
- template<typename Integers , size_t deg>
 using **tan** = **taylor**< Integers, internal::tan_coeff, deg >
 tan(x) *tangent*
- template<typename Integers , size_t deg>
 using **tanh** = **taylor**< Integers, internal::tanh_coeff, deg >
 tanh(x) *hyperbolic tangent*
- using **PI_fraction** = **ContinuedFraction**< 3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1 >
- using **E_fraction** = **ContinuedFraction**< 2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1 >
 approximation of e
- using **SQRT2_fraction** = **ContinuedFraction**< 1, 2 >
 approximation of $\sqrt{2}$
- using **SQRT3_fraction** = **ContinuedFraction**< 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 >
 approximation of

Functions

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

Variables

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

6.1.1 Detailed Description

main namespace for all publicly exposed types or functions

6.1.2 Typedef Documentation

6.1.2.1 abs_t

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

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

Template Parameters

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

6.1.2.2 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 FranctionField<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 $\sqrt{2}$

6.1.2.42 Sqrt3_fraction

```
using aerobus::Sqrt3_fraction = typedef ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2>
```

approximation of

6.1.2.43 stirling_1_signed_t

```
template<typename T , int n, int k>
using aerobus::stirling_1_signed_t = typedef typename internal::stirling_1_helper<T, n, k>::type
```

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

Template Parameters

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

6.1.2.44 stirling_1_unsigned_t

```
template<typename T , int n, int k>
using aerobus::stirling_1_unsigned_t = typedef abs_t<typename internal::stirling_1_helper<T, n, k>::type>
```

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

Template Parameters

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

6.1.2.45 stirling_2_t

```
template<typename T , int n, int k>
using aerobus::stirling_2_t = typedef typename internal::stirling_2_helper<T, n, k>::type
```

Stirling number of second king – as types.

Template Parameters

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

6.1.2.46 sub_t

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

generic subtraction

Template Parameters

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

6.1.2.47 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

6.1.2.48 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

6.1.2.49 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>	

6.1.2.50 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>	
----------------	--

6.1.2.51 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 **AbelHelper**
- struct **AllOneHelper**
- struct **AllOneHelper**< 0, I >
- 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 **FloatLayout**
- struct **FloatLayout**< double >
- struct **FloatLayout**< float >
- struct **FloatLayout**< long double >
- 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 **fma_helper**< long double >
- 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, I >
- 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**
- struct **split_h**
- struct **split_h**< 0, L1, L2 >
- struct **staticcast**
- struct **stirling_1_helper**
- struct **stirling_1_helper**< T, 0, 0 >
- struct **stirling_1_helper**< T, 0, n, std::enable_if_t<(n > 0)> >
- struct **stirling_1_helper**< T, n, 0, std::enable_if_t<(n > 0)> >

- struct **stirling_1_helper**< T, n, k, std::enable_if_t<(k > 0) &&(n > 0)> >
- struct **stirling_2_helper**
- struct **stirling_2_helper**< T, 0, n, std::enable_if_t<(n > 0)> >
- struct **stirling_2_helper**< T, n, 0, std::enable_if_t<(n > 0)> >
- struct **stirling_2_helper**< T, n, k, std::enable_if_t<(k > 0) &&(n > 0) &&(k < n)> >
- struct **stirling_2_helper**< T, n, n, std::enable_if_t<(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 **touchard_coeff**
- 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

Enumerations

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

6.3.1 Detailed Description

families of well known polynomials such as Hermite or Bernstein

6.3.2 Enumeration Type Documentation

6.3.2.1 hermite_kind

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

Enumerator

probabilist	
physicist	

Chapter 7

Concept Documentation

7.1 aerobus::IsEuclideanDomain Concept Reference

Concept to express R is an euclidean domain.

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

7.1.1 Concept definition

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

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

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

7.1.2 Detailed Description

Concept to express R is an euclidean domain.

7.2 aerobus::IsField Concept Reference

Concept to express R is a field.

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

7.2.1 Concept definition

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

7.2.2 Detailed Description

Concept to express R is a field.

7.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring.

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

7.3.1 Concept definition

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

7.3.2 Detailed Description

Concept to express R is a Ring.

Chapter 8

Class Documentation

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

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

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

- [src/aerobus.h](#)

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

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

Public Types

- using `type` = typename Ring::zero

8.2.1 Member Typedef Documentation

8.2.1.1 `type`

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

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

- [src/aerobus.h](#)

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

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

Public Types

- using [type](#) = [aN](#)

8.3.1 Member Typedef Documentation

8.3.1.1 type

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

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

- [src/aerobus.h](#)

8.4 aerobus::ContinuedFraction< values > Struct Template Reference

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

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

8.4.1 Detailed Description

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

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

Template Parameters

<i>...values</i>	are <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::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner< index, ghost > Struct Template Reference

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

Static Public Member Functions

- static [INLINE DEVICE](#) void [func](#) (arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmetic↵
Type *r)

8.8.1 Member Function Documentation

8.8.1.1 func()

```
template<typename Ring >
template<typename arithmeticType , typename P >
template<int64_t index, int ghost>
static INLINE DEVICE void aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P
>::EFTHorner< index, ghost >::func (
    arithmeticType x,
    arithmeticType * pi,
    arithmeticType * sigma,
    arithmeticType * r ) [inline], [static]
```

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

- [src/aerobus.h](#)

8.9 aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner<-1, ghost > Struct Template Reference

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

Static Public Member Functions

- static **INLINE** **DEVICE** void **func** (arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType *r)

8.9.1 Member Function Documentation

8.9.1.1 func()

```
template<typename Ring >
template<typename arithmeticType , typename P >
template<int ghost>
static INLINE DEVICE void aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P
>::EFTHorner<-1, ghost >::func (
    arithmeticType x,
    arithmeticType * pi,
    arithmeticType * sigma,
    arithmeticType * r ) [inline], [static]
```

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

- [src/aerobus.h](#)

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

embedding - struct forward declaration

8.10.1 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](#)

8.11 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

8.11.1 Detailed Description

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

8.11.2 Member Typedef Documentation

8.11.2.1 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](#)

8.12 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

8.12.1 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

8.12.2 Member Typedef Documentation

8.12.2.1 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](#)

8.13 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

8.13.1 Detailed Description

embeds q32 into q64

8.13.2 Member Typedef Documentation

8.13.2.1 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](#)

8.14 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.

8.14.1 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

8.14.2 Member Typedef Documentation

8.14.2.1 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](#)

8.15 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.

8.15.1 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
-------------	---

8.15.2 Member Typedef Documentation

8.15.2.1 type

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

`FractionField<Ring>` representation of v.

Template Parameters

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

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

- `src/aerobus.h`

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

embeds zpz values into [i32](#)

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

Public Types

- `template<typename val >`
`using type = i32::val< val::v >`
the i32 representation of val

8.16.1 Detailed Description

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

embeds zpz values into i32

Template Parameters

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

8.16.2 Member Typedef Documentation

8.16.2.1 type

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

the i32 representation of val

Template Parameters

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

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

- `src/aerobus.h`

8.17 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](#) >

8.17.1 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](#)

8.18 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

8.18.1 Detailed Description

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

Examples

[examples/compensated_horner.cpp](#).

8.18.2 Member Typedef Documentation

8.18.2.1 add_t

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

addition operator yields $v1 + v2$

Template Parameters

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

8.18.2.2 div_t

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

division operator yields $v1 / v2$

Template Parameters

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

8.18.2.3 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

8.18.2.4 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

8.18.2.5 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

8.18.2.6 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>	
----------	--

8.18.2.7 inject_ring_t

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

8.18.2.8 inner_type

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

8.18.2.9 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

8.18.2.10 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

8.18.2.11 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

8.18.2.12 one

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

constant one

8.18.2.13 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
----------	--------------------------------

8.18.2.14 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

8.18.2.15 zero

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

constant zero

8.18.3 Member Data Documentation

8.18.3.1 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>	

8.18.3.2 is_euclidean_domain

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

integers are an euclidean domain

8.18.3.3 is_field

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

integers are not a field

8.18.3.4 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](#)

8.19 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

8.19.1 Detailed Description

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

8.19.2 Member Typedef Documentation

8.19.2.1 [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

8.19.2.2 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

8.19.2.3 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

8.19.2.4 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

8.19.2.5 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

8.19.2.6 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>	
----------------	--

8.19.2.7 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
----------------	--------------------------------

8.19.2.8 inner_type

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

type of represented values

8.19.2.9 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

8.19.2.10 mod_t

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

modulus operator

Template Parameters

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

8.19.2.11 mul_t

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

multiplication operator

Template Parameters

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

8.19.2.12 one

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

constant one

8.19.2.13 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

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

8.19.2.14 sub_t

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

substraction operator

Template Parameters

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

8.19.2.15 zero

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

constant zero

8.19.3 Member Data Documentation

8.19.3.1 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

8.19.3.2 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

8.19.3.3 is_euclidean_domain

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

integers are an euclidean domain

8.19.3.4 is_field

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

integers are not a field

8.19.3.5 lt_v

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

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

Template Parameters

$v1$: an element of aerobus::i64::val
$v2$: an element of aerobus::i64::val

8.19.3.6 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](#)

8.20 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 >`

8.20.1 Member Typedef Documentation

8.20.1.1 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](#)

8.21 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`

8.21.1 Member Typedef Documentation

8.21.1.1 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](#)

8.22 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

8.22.1 Detailed Description

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

checks if n is prime

Template Parameters

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

8.22.2 Member Data Documentation

8.22.2.1 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](#)

8.23 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

8.23.1 Detailed Description

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

polynomial with coefficients in Ring Ring must be an integral domain

Examples

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

8.23.2 Member Typedef Documentation

8.23.2.1 add_t

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

adds two polynomials

Template Parameters

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

8.23.2.2 derive_t

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

derivation operator

Template Parameters

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

8.23.2.3 div_t

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

division operator

Template Parameters

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

8.23.2.4 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>	

8.23.2.5 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>	

8.23.2.6 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>	

8.23.2.7 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>	
----------	--

8.23.2.8 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>	
----------	--

8.23.2.9 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>	

8.23.2.10 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>	

8.23.2.11 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>	

8.23.2.12 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>	

8.23.2.13 one

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

constant one

8.23.2.14 pos_t

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

checks for positivity ($an > 0$)

Template Parameters

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

8.23.2.15 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>	
----------	--

8.23.2.16 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>	

8.23.2.17 X

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

generator

8.23.2.18 zero

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

constant zero

8.23.3 Member Data Documentation

8.23.3.1 is_euclidean_domain

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

8.23.3.2 is_field

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

8.23.3.3 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](#)

8.24 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

8.24.1 Detailed Description

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

removes types from head of the list

8.24.2 Member Typedef Documentation

8.24.2.1 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

8.24.2.2 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](#)

8.25 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

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

8.25.2 Member Typedef Documentation

8.25.2.1 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

8.25.2.2 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

8.25.2.3 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

8.25.2.4 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
----------	--------------------------------------

8.25.2.5 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
----------	-----------------

8.25.2.6 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

8.25.2.7 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

8.25.2.8 one

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

one

8.25.2.9 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
-----------	--------------------------

8.25.2.10 zero

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

zero value

8.25.3 Member Data Documentation

8.25.3.1 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

8.25.3.2 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

8.25.3.3 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](#)

8.26 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

8.26.1 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>	
--------------	--

8.26.2 Member Typedef Documentation

8.26.2.1 head

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

8.26.2.2 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

8.27 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

8.27.1 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
--------------	---

8.27.2 Member Typedef Documentation

8.27.2.1 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>	
--------------	--

8.27.2.2 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>	
----------	--

8.27.2.3 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>	

8.27.2.4 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>	
----------	--

8.27.2.5 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>	
----------	--

8.27.2.6 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>	
--------------	--

8.27.3 Member Data Documentation

8.27.3.1 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](#)

8.28 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

8.28.1 Detailed Description

specialization for empty type list

8.28.2 Member Typedef Documentation

8.28.2.1 concat

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

8.28.2.2 insert

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

8.28.2.3 push_back

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

8.28.2.4 push_front

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

8.28.3 Member Data Documentation

8.28.3.1 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

8.29 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

8.29.1 Detailed Description

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

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

Template Parameters

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

8.29.2 Member Typedef Documentation

8.29.2.1 enclosing_type

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

Enclosing ring type.

8.29.2.2 is_zero_t

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

is value zero

8.29.3 Member Function Documentation

8.29.3.1 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
------------------	--------------------

8.29.3.2 to_string()

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

string representation of value

8.29.4 Member Data Documentation

8.29.4.1 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](#)

8.30 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

8.30.1 Detailed Description

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

values in [i64](#)

Template Parameters

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

Examples

[examples/compensated_horner.cpp](#).

8.30.2 Member Typedef Documentation

8.30.2.1 enclosing_type

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

enclosing ring type

8.30.2.2 inner_type

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

type of represented values

8.30.2.3 is_zero_t

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

is value zero

8.30.3 Member Function Documentation

8.30.3.1 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)
------------------	----------------------

8.30.3.2 to_string()

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

string representation

8.30.4 Member Data Documentation

8.30.4.1 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

8.31 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
- template<typename arithmeticType >
static [DEVICE INLINED](#) arithmeticType [compensated_eval](#) (const arithmeticType &x)
Evaluate polynomial on x using compensated horner scheme.

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

8.31.1 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

Examples

[examples/compensated_horner.cpp](#).

8.31.2 Member Typedef Documentation

8.31.2.1 aN

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

heavy weight coefficient (non zero)

8.31.2.2 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>	
--------------	--

8.31.2.3 enclosing_type

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

enclosing ring type
```

8.31.2.4 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
```

8.31.2.5 ring_type

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

ring coefficients live in
```

8.31.2.6 strip

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

remove largest coefficient
```

8.31.2.7 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>
```

8.31.3 Member Function Documentation

8.31.3.1 compensated_eval()

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

Evaluate polynomial on x using compensated horner scheme.

This is twice as accurate as simple eval (horner) but cannot be constexpr

Please note this makes no sense on integer types as arithmetic on integers is exact in IEEE

WARNING : this does not work with gcc with -O3 optimization level because gcc does illegal stuff with floating point arithmetic

Template Parameters

<i>arithmeticType</i>	float for example
-----------------------	-------------------

Parameters

x	
---	--

8.31.3.2 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

x	value
---	-------

Returns

$P(x)$

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

8.31.4 Member Data Documentation

8.31.4.1 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

8.31.4.2 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](#)

8.32 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 > >

8.32.1 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'
---	---------------------

8.32.2 Member Typedef Documentation

8.32.2.1 raw_t

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

8.32.2.2 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](#)

8.33 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

8.33.1 Detailed Description

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

values in zpz

Template Parameters

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

8.33.2 Member Typedef Documentation

8.33.2.1 enclosing_type

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

enclosing ring type

8.33.2.2 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

8.33.3 Member Function Documentation

8.33.3.1 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
------------------	-----------------------------------

8.33.3.2 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

8.33.4 Member Data Documentation**8.33.4.1 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

8.33.4.2 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](#)

8.34 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)
- template<typename arithmeticType >
static [DEVICE INLINED](#) arithmeticType [compensated_eval](#) (const arithmeticType &x)

Static Public Attributes

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

8.34.1 Detailed Description

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

specialization for constants

Template Parameters

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

8.34.2 Member Typedef Documentation

8.34.2.1 aN

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

8.34.2.2 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
```

8.34.2.3 enclosing_type

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

enclosing ring type

8.34.2.4 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>
```

8.34.2.5 ring_type

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

ring coefficients live in

8.34.2.6 strip

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

8.34.2.7 value_at_t

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

8.34.3 Member Function Documentation

8.34.3.1 compensated_eval()

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

8.34.3.2 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]
```

8.34.3.3 to_string()

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

8.34.4 Member Data Documentation

8.34.4.1 degree

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

degree

8.34.4.2 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](#)

8.35 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 mpz
- 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)

8.35.1 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](#).

8.35.2 Member Typedef Documentation

8.35.2.1 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

8.35.2.2 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

8.35.2.3 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

8.35.2.4 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

8.35.2.5 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

8.35.2.6 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

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

8.35.2.7 inner_type

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

underlying type for values

8.35.2.8 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

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

8.35.2.9 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

8.35.2.10 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

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

8.35.2.11 one

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

one value

8.35.2.12 pos_t

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

positivity operator (type)

Template Parameters

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

8.35.2.13 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

8.35.2.14 zero

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

zero value

8.35.3 Member Data Documentation

8.35.3.1 eq_v

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

equality operator (booleanvalue)

Template Parameters

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

8.35.3.2 gt_v

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

strictly greater operator (booleanvalue)

Template Parameters

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

8.35.3.3 is_euclidean_domain

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

always true

8.35.3.4 is_field

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

true iff p is prime

8.35.3.5 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

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

8.35.3.6 pos_v

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

positivity operator (boolean value)

Template Parameters

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

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

- [src/aerobus.h](#)

Chapter 9

File Documentation

9.1 README.md File Reference

9.2 src/aerobus.h File Reference

```
#include <cstdint>
#include <cstddef>
#include <cstring>
#include <type_traits>
#include <utility>
#include <algorithm>
#include <functional>
#include <string>
#include <concepts>
#include <array>
Include dependency graph for aerobus.h:
```

9.3 aerobus.h

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

```
00001 // -*- lsst-c++ -*-
00002 #ifndef __INC_AEROBUS__ // NOLINT
00003 #define __INC_AEROBUS__
00004
00005 #include <cstdint>
00006 #include <cstddef>
00007 #include <cstring>
00008 #include <type_traits>
00009 #include <utility>
00010 #include <algorithm>
00011 #include <functional>
00012 #include <string>
00013 #include <concepts> // NOLINT
00014 #include <array>
00015 #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 > 16U) & 0x8000U);
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<long double> {
00224         static constexpr INLINED_DEVICE long double eval(
00225             const long double x, const long double y, const long double z) {
00226             return x * y + z;
00227         }
00228     };
00229
00230     template<>
00231     struct fma_helper<float> {
00232         static constexpr INLINED_DEVICE float eval(const float x, const float y, const float z) {
00233             return x * y + z;
00234         }
00235     };
00236
00237     template<>
00238     struct fma_helper<int32_t> {
00239         static constexpr INLINED_DEVICE int16_t eval(const int16_t x, const int16_t y, const
int16_t z) {
00240             return x * y + z;
00241         }
00242     };
00243
00244     template<>
00245     struct fma_helper<int16_t> {
00246         static constexpr INLINED_DEVICE int32_t eval(const int32_t x, const int32_t y, const
int32_t z) {
00247             return x * y + z;
00248         }
00249     };
00250
00251     template<>
00252     struct fma_helper<int64_t> {
00253         static constexpr INLINED_DEVICE int64_t eval(const int64_t x, const int64_t y, const
int64_t z) {
00254             return x * y + z;
00255         }
00256     };
00257
00258     #ifdef WITH_CUDA_FP16
00259     template<>
00260     struct fma_helper<__half> {
00261         static constexpr INLINED_DEVICE __half eval(const __half x, const __half y, const __half
z) {
00262             #ifdef __CUDA_ARCH__
00263                 return __hfma(x, y, z);
00264             #else
00265                 return x * y + z;
00266             #endif
00267         }
00268     };
00269     template<>
00270     struct fma_helper<__half2> {
00271         static constexpr INLINED_DEVICE __half2 eval(const __half2 x, const __half2 y, const
__half2 z) {
00272             #ifdef __CUDA_ARCH__
00273                 return __hfma2(x, y, z);
00274             #else
00275                 return x * y + z;
00276             #endif
00277         }
00278     };
00279     #endif
00280 } // namespace internal
00281 } // namespace aerobus
00282
00283 // compensated horner utilities
00284 namespace aerobus {
00285     namespace internal {
00286         template <typename T>
00287         struct FloatLayout;
00288
00289         #ifdef _MSC_VER
00290         template <>
00291         struct FloatLayout<long double> {
00292             static constexpr uint8_t exponent = 11;
00293             static constexpr uint8_t mantissa = 53;
00294             static constexpr uint8_t r = 27; // ceil(mantissa/2)
00295         };

```



```

00296     #else
00297     template <>
00298     struct FloatLayout<long double> {
00299         static constexpr uint8_t exponent = 15;
00300         static constexpr uint8_t mantissa = 63;
00301         static constexpr uint8_t r = 32; // ceil(mantissa/2)
00302         static constexpr long double shift = (1LL « r) + 1;
00303     };
00304     #endif
00305
00306     template <>
00307     struct FloatLayout<double> {
00308         static constexpr uint8_t exponent = 11;
00309         static constexpr uint8_t mantissa = 53;
00310         static constexpr uint8_t r = 27; // ceil(mantissa/2)
00311         static constexpr double shift = (1LL « r) + 1;
00312     };
00313
00314     template <>
00315     struct FloatLayout<float> {
00316         static constexpr uint8_t exponent = 8;
00317         static constexpr uint8_t mantissa = 24;
00318         static constexpr uint8_t r = 11; // ceil(mantissa/2)
00319         static constexpr float shift = (1 « r) + 1;
00320     };
00321
00322     template<typename T>
00323     struct Split {
00324         static constexpr INLINED_DEVICE void func(T a, T *x, T *y) {
00325             T z = a * FloatLayout<T>::shift;
00326             *x = z - (z - a);
00327             *y = a - *x;
00328         }
00329     };
00330
00331     #ifdef WITH_CUDA_FP16
00332     template<>
00333     struct Split<__half> {
00334         static constexpr INLINED_DEVICE void func(__half a, __half *x, __half *y) {
00335             __half z = a * __half_raw(0x5280); // TODO(JeWaVe): check this value
00336             *x = z - (z - a);
00337             *y = a - *x;
00338         }
00339     };
00340
00341     template<>
00342     struct Split<__half2> {
00343         static constexpr INLINED_DEVICE void func(__half2 a, __half2 *x, __half2 *y) {
00344             __half2 z = a * __half2(__half_raw(0x5280), __half_raw(0x5280)); // TODO(JeWaVe):
00345             check this value
00346             *x = z - (z - a);
00347             *y = a - *x;
00348         }
00349     };
00350     #endif
00351
00352     template<typename T>
00353     static constexpr INLINED_DEVICE void two_sum(T a, T b, T *x, T *y) {
00354         *x = a + b;
00355         T z = *x - a;
00356         *y = (a - (*x - z)) + (b - z);
00357     }
00358
00359     template<typename T>
00360     static constexpr INLINED_DEVICE void two_prod(T a, T b, T *x, T *y) {
00361         *x = a * b;
00362         #ifdef __clang__
00363         *y = fma_helper<T>::eval(a, b, -*x);
00364         #else
00365         T ah, al, bh, bl;
00366         Split<T>::func(a, &ah, &al);
00367         Split<T>::func(b, &bh, &bl);
00368         *y = al * bl - ((*x - ah * bh) - al * bh) - ah * bl;
00369         #endif
00370     }
00371
00372     template<typename T, size_t N>
00373     static INLINED_DEVICE T horner(T *p1, T *p2, T x) {
00374         T r = p1[0] + p2[0];
00375         for (int64_t i = N - 1; i >= 0; --i) {
00376             r = r * x + p1[N - i] + p2[N - i];
00377         }
00378         return r;
00379     }
00380 } // namespace internal
00381 } // namespace aerobus

```

```

00382
00383 // utilities
00384 namespace aerobus {
00385     namespace internal {
00386         template<template<typename...> typename TT, typename T>
00387             struct is_instantiation_of : std::false_type { };
00388
00389         template<template<typename...> typename TT, typename... Ts>
00390             struct is_instantiation_of<TT, TT<Ts...> : std::true_type { };
00391
00392         template<template<typename...> typename TT, typename T>
00393             inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00394
00395         template<int64_t i, typename T, typename... Ts>
00396             struct type_at {
00397                 static_assert(i < sizeof...(Ts) + 1, "index out of range");
00398                 using type = typename type_at<i - 1, Ts...>::type;
00399             };
00400
00401         template<typename T, typename... Ts> struct type_at<0, T, Ts...> {
00402             using type = T;
00403         };
00404
00405         template<size_t i, typename... Ts>
00406             using type_at_t = typename type_at<i, Ts...>::type;
00407
00408
00409         template<size_t n, size_t i, typename E = void>
00410             struct _is_prime {};
00411
00412         template<size_t i>
00413             struct _is_prime<0, i> {
00414                 static constexpr bool value = false;
00415             };
00416
00417         template<size_t i>
00418             struct _is_prime<1, i> {
00419                 static constexpr bool value = false;
00420             };
00421
00422         template<size_t i>
00423             struct _is_prime<2, i> {
00424                 static constexpr bool value = true;
00425             };
00426
00427         template<size_t i>
00428             struct _is_prime<3, i> {
00429                 static constexpr bool value = true;
00430             };
00431
00432         template<size_t i>
00433             struct _is_prime<5, i> {
00434                 static constexpr bool value = true;
00435             };
00436
00437         template<size_t i>
00438             struct _is_prime<7, i> {
00439                 static constexpr bool value = true;
00440             };
00441
00442         template<size_t n, size_t i>
00443             struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)> {
00444                 static constexpr bool value = false;
00445             };
00446
00447         template<size_t n, size_t i>
00448             struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)> {
00449                 static constexpr bool value = false;
00450             };
00451
00452         template<size_t n, size_t i>
00453             struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)> {
00454                 static constexpr bool value = true;
00455             };
00456
00457         template<size_t n, size_t i>
00458             struct _is_prime<n, i, std::enable_if_t<(
00459                 n % i == 0 &&
00460                 n >= 9 &&
00461                 n % 3 != 0 &&
00462                 n % 2 != 0 &&
00463                 i * i > n)> {
00464                 static constexpr bool value = true;
00465             };
00466
00467         template<size_t n, size_t i>
00468             struct _is_prime<n, i, std::enable_if_t<(

```

```

00469         n % (i+2) == 0 &&
00470         n >= 9 &&
00471         n % 3 != 0 &&
00472         n % 2 != 0 &&
00473         i * i <= n)) {
00474     static constexpr bool value = true;
00475 };
00476
00477 template<size_t n, size_t i>
00478 struct _is_prime<n, i, std::enable_if_t<(
00479     n % (i+2) != 0 &&
00480     n % i != 0 &&
00481     n >= 9 &&
00482     n % 3 != 0 &&
00483     n % 2 != 0 &&
00484     (i * i <= n))> {
00485     static constexpr bool value = _is_prime<n, i+6>::value;
00486 };
00487 } // namespace internal
00488
00491 template<size_t n>
00492 struct is_prime {
00493     static constexpr bool value = internal::_is_prime<n, 5>::value;
00494 };
00495
00496 template<size_t n>
00500 static constexpr bool is_prime_v = is_prime<n>::value;
00501
00502 // gcd
00503 namespace internal {
00504     template <std::size_t... Is>
00505     constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&
00506         -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{}));
00507
00508     template <std::size_t N>
00509     using make_index_sequence_reverse
00510         = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00511
00512     template<typename Ring, typename E = void>
00513     struct gcd;
00514
00515     template<typename Ring>
00516     struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain> {
00517         template<typename A, typename B, typename E = void>
00518         struct gcd_helper {};
00519
00520         // B = 0, A > 0
00521         template<typename A, typename B>
00522         struct gcd_helper<A, B, std::enable_if_t<
00523             ((B::is_zero_t::value) &&
00524              (Ring::template gt_t<A, typename Ring::zero>::value))> {
00525             using type = A;
00526         };
00527
00528         // B = 0, A < 0
00529         template<typename A, typename B>
00530         struct gcd_helper<A, B, std::enable_if_t<
00531             ((B::is_zero_t::value) &&
00532              !(Ring::template gt_t<A, typename Ring::zero>::value))> {
00533             using type = typename Ring::template sub_t<typename Ring::zero, A>;
00534         };
00535
00536         // B != 0
00537         template<typename A, typename B>
00538         struct gcd_helper<A, B, std::enable_if_t<
00539             (!B::is_zero_t::value)
00540             > {
00541             private: // NOLINT
00542                 // A / B
00543                 using k = typename Ring::template div_t<A, B>;
00544                 // A - (A/B)*B = A % B
00545                 using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B>;
00546
00547             public:
00548                 using type = typename gcd_helper<B, m>::type;
00549         };
00550
00551         template<typename A, typename B>
00552         using type = typename gcd_helper<A, B>::type;
00553     };
00554 } // namespace internal
00555
00556 // vadd and vmul
00557 namespace internal {
00558     template<typename... vals>
00559     struct vmul {};
00560 }

```

```

00567     template<typename v1, typename... vals>
00568     struct vmul<v1, vals...> {
00569         using type = typename v1::enclosing_type::template mul_t<v1, typename
vmul<vals...>::type>;
00570     };
00571
00572     template<typename v1>
00573     struct vmul<v1> {
00574         using type = v1;
00575     };
00576
00577     template<typename... vals>
00578     struct vadd {};
00579
00580     template<typename v1, typename... vals>
00581     struct vadd<v1, vals...> {
00582         using type = typename v1::enclosing_type::template add_t<v1, typename
vadd<vals...>::type>;
00583     };
00584
00585     template<typename v1>
00586     struct vadd<v1> {
00587         using type = v1;
00588     };
00589 } // namespace internal
00590
00593 template<typename T, typename A, typename B>
00594 using gcd_t = typename internal::gcd<T>::template type<A, B>;
00595
00599 template<typename... vals>
00600 using vadd_t = typename internal::vadd<vals...>::type;
00601
00605 template<typename... vals>
00606 using vmul_t = typename internal::vmul<vals...>::type;
00607
00611 template<typename val>
00612 requires IsEuclideanDomain<typename val::enclosing_type>
00613 using abs_t = std::conditional_t<
00614     val::enclosing_type::template pos_v<val>,
00615     val, typename val::enclosing_type::template
sub_t<typename val::enclosing_type::zero, val>>;
00616 } // namespace aerobus
00617
00618 // embedding
00619 namespace aerobus {
00624     template<typename Small, typename Large, typename E = void>
00625     struct Embed;
00626 } // namespace aerobus
00627
00628 namespace aerobus {
00633     template<typename Ring, typename X>
00634     requires IsRing<Ring>
00635     struct Quotient {
00638         template <typename V>
00639         struct val {
00640             public:
00641                 using raw_t = V;
00642                 using type = abs_t<typename Ring::template mod_t<V, X>>;
00643         };
00644
00646         using zero = val<typename Ring::zero>;
00647
00649         using one = val<typename Ring::one>;
00650
00654         template<typename v1, typename v2>
00655         using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00656
00660         template<typename v1, typename v2>
00661         using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00662
00666         template<typename v1, typename v2>
00667         using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00668
00672         template<typename v1, typename v2>
00673         using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00674
00678         template<typename v1, typename v2>
00679         using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00680
00684         template<typename v1, typename v2>
00685         static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00686
00690         template<typename v1>
00691         using pos_t = std::true_type;
00692
00696         template<typename v>
00697         static constexpr bool pos_v = pos_t<v>::value;

```

```

00698
00700     static constexpr bool is_euclidean_domain = true;
00701
00705     template<auto x>
00706     using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00707
00711     template<typename v>
00712     using inject_ring_t = val<v>;
00713 };
00714
00718     template<typename Ring, typename X>
00719     struct Embed<Quotient<Ring, X>, Ring> {
00722         template<typename val>
00723         using type = typename val::raw_t;
00724     };
00725 } // namespace aerobus
00726
00727 // type_list
00728 namespace aerobus {
00730     template <typename... Ts>
00731     struct type_list;
00732
00733     namespace internal {
00734         template <typename T, typename... Us>
00735         struct pop_front_h {
00736             using tail = type_list<Us...>;
00737             using head = T;
00738         };
00739
00740         template <size_t index, typename L1, typename L2>
00741         struct split_h {
00742             private:
00743                 static_assert(index <= L2::length, "index ouf of bounds");
00744                 using a = typename L2::pop_front::type;
00745                 using b = typename L2::pop_front::tail;
00746                 using c = typename L1::template push_back<a>;
00747
00748             public:
00749                 using head = typename split_h<index - 1, c, b>::head;
00750                 using tail = typename split_h<index - 1, c, b>::tail;
00751         };
00752
00753         template <typename L1, typename L2>
00754         struct split_h<0, L1, L2> {
00755             using head = L1;
00756             using tail = L2;
00757         };
00758
00759         template <size_t index, typename L, typename T>
00760         struct insert_h {
00761             static_assert(index <= L::length, "index ouf of bounds");
00762             using s = typename L::template split<index>;
00763             using left = typename s::head;
00764             using right = typename s::tail;
00765             using ll = typename left::template push_back<T>;
00766             using type = typename ll::template concat<right>;
00767         };
00768
00769         template <size_t index, typename L>
00770         struct remove_h {
00771             using s = typename L::template split<index>;
00772             using left = typename s::head;
00773             using right = typename s::tail;
00774             using rr = typename right::pop_front::tail;
00775             using type = typename left::template concat<rr>;
00776         };
00777     } // namespace internal
00778
00781     template <typename... Ts>
00782     struct type_list {
00783     private:
00784         template <typename T>
00785         struct concat_h;
00786
00787         template <typename... Us>
00788         struct concat_h<type_list<Us...> {
00789             using type = type_list<Ts..., Us...>;
00790         };
00791
00792     public:
00794         static constexpr size_t length = sizeof...(Ts);
00795
00798         template <typename T>
00799         using push_front = type_list<T, Ts...>;
00800
00803         template <size_t index>
00804         using at = internal::type_at_t<index, Ts...>;

```

```

00805
00807     struct pop_front {
00809         using type = typename internal::pop_front_h<Ts...>::head;
00811         using tail = typename internal::pop_front_h<Ts...>::tail;
00812     };
00813
00816     template <typename T>
00817     using push_back = type_list<Ts..., T>;
00818
00821     template <typename U>
00822     using concat = typename concat_h<U>::type;
00823
00826     template <size_t index>
00827     struct split {
00828     private:
00829         using inner = internal::split_h<index, type_list<>, type_list<Ts...>;
00830
00831     public:
00832         using head = typename inner::head;
00833         using tail = typename inner::tail;
00834     };
00835
00839     template <typename T, size_t index>
00840     using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00841
00844     template <size_t index>
00845     using remove = typename internal::remove_h<index, type_list<Ts...>::type;
00846 };
00847
00849 template <>
00850 struct type_list<> {
00851     static constexpr size_t length = 0;
00852
00853     template <typename T>
00854     using push_front = type_list<T>;
00855
00856     template <typename T>
00857     using push_back = type_list<T>;
00858
00859     template <typename U>
00860     using concat = U;
00861
00862     // TODO(jewave): assert index == 0
00863     template <typename T, size_t index>
00864     using insert = type_list<T>;
00865 };
00866 } // namespace aerobus
00867
00868 // i16
00869 #ifdef WITH_CUDA_FP16
00870 // i16
00871 namespace aerobus {
00872     struct i16 {
00873         using inner_type = int16_t;
00874         template<int16_t x>
00875         struct val {
00876             using enclosing_type = i16;
00877             static constexpr int16_t v = x;
00878
00879             template<typename valueType>
00880             static constexpr INLINED_DEVICE valueType get() {
00881                 return internal::template int16_convert_helper<valueType, x>::value();
00882             }
00883
00884             using is_zero_t = std::bool_constant<x == 0>;
00885
00886             static std::string to_string() {
00887                 return std::to_string(x);
00888             }
00889         };
00890
00891         using zero = val<0>;
00892         using one = val<1>;
00893         static constexpr bool is_field = false;
00894         static constexpr bool is_euclidean_domain = true;
00895         template<auto x>
00896         using inject_constant_t = val<static_cast<int16_t>(x)>;
00897
00898         template<typename v>
00899         using inject_ring_t = v;
00900
00901     private:
00902         template<typename v1, typename v2>
00903         struct add {
00904             using type = val<v1::v + v2::v>;
00905         };
00906     };
00907 }

```

```

00922     template<typename v1, typename v2>
00923     struct sub {
00924         using type = val<v1::v - v2::v>;
00925     };
00926
00927     template<typename v1, typename v2>
00928     struct mul {
00929         using type = val<v1::v* v2::v>;
00930     };
00931
00932     template<typename v1, typename v2>
00933     struct div {
00934         using type = val<v1::v / v2::v>;
00935     };
00936
00937     template<typename v1, typename v2>
00938     struct remainder {
00939         using type = val<v1::v % v2::v>;
00940     };
00941
00942     template<typename v1, typename v2>
00943     struct gt {
00944         using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00945     };
00946
00947     template<typename v1, typename v2>
00948     struct lt {
00949         using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00950     };
00951
00952     template<typename v1, typename v2>
00953     struct eq {
00954         using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00955     };
00956
00957     template<typename v1>
00958     struct pos {
00959         using type = std::bool_constant<(v1::v > 0)>;
00960     };
00961
00962     public:
00963     template<typename v1, typename v2>
00964     using add_t = typename add<v1, v2>::type;
00965
00966     template<typename v1, typename v2>
00967     using sub_t = typename sub<v1, v2>::type;
00968
00969     template<typename v1, typename v2>
00970     using mul_t = typename mul<v1, v2>::type;
00971
00972     template<typename v1, typename v2>
00973     using div_t = typename div<v1, v2>::type;
00974
00975     template<typename v1, typename v2>
00976     using mod_t = typename remainder<v1, v2>::type;
00977
00978     template<typename v1, typename v2>
00979     using gt_t = typename gt<v1, v2>::type;
00980
00981     template<typename v1, typename v2>
00982     using lt_t = typename lt<v1, v2>::type;
00983
00984     template<typename v1, typename v2>
00985     using eq_t = typename eq<v1, v2>::type;
00986
00987     template<typename v1, typename v2>
00988     static constexpr bool eq_v = eq_t<v1, v2>::value;
00989
00990     template<typename v1, typename v2>
00991     using gcd_t = gcd_t<i16, v1, v2>;
00992
00993     template<typename v>
00994     using pos_t = typename pos<v>::type;
00995
00996     template<typename v>
00997     static constexpr bool pos_v = pos_t<v>::value;
00998 };
00999 } // namespace aerobus
01000 #endif
01001
01002 // i32
01003 namespace aerobus {
01004     struct i32 {
01005         using inner_type = int32_t;
01006         template<int32_t x>
01007         struct val {
01008             using enclosing_type = i32;

```

```

01059         static constexpr int32_t v = x;
01060
01063         template<typename valueType>
01064         static constexpr DEVICE valueType get() {
01065             return static_cast<valueType>(x);
01066         }
01067
01069         using is_zero_t = std::bool_constant<x == 0>;
01070
01072         static std::string to_string() {
01073             return std::to_string(x);
01074         }
01075     };
01076
01078     using zero = val<0>;
01080     using one = val<1>;
01082     static constexpr bool is_field = false;
01084     static constexpr bool is_euclidean_domain = true;
01087     template<auto x>
01088     using inject_constant_t = val<static_cast<int32_t>(x)>;
01089
01090     template<typename v>
01091     using inject_ring_t = v;
01092
01093 private:
01094     template<typename v1, typename v2>
01095     struct add {
01096         using type = val<v1::v + v2::v>;
01097     };
01098
01099     template<typename v1, typename v2>
01100     struct sub {
01101         using type = val<v1::v - v2::v>;
01102     };
01103
01104     template<typename v1, typename v2>
01105     struct mul {
01106         using type = val<v1::v * v2::v>;
01107     };
01108
01109     template<typename v1, typename v2>
01110     struct div {
01111         using type = val<v1::v / v2::v>;
01112     };
01113
01114     template<typename v1, typename v2>
01115     struct remainder {
01116         using type = val<v1::v % v2::v>;
01117     };
01118
01119     template<typename v1, typename v2>
01120     struct gt {
01121         using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01122     };
01123
01124     template<typename v1, typename v2>
01125     struct lt {
01126         using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01127     };
01128
01129     template<typename v1, typename v2>
01130     struct eq {
01131         using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01132     };
01133
01134     template<typename v1>
01135     struct pos {
01136         using type = std::bool_constant<(v1::v > 0)>;
01137     };
01138
01139 public:
01144     template<typename v1, typename v2>
01145     using add_t = typename add<v1, v2>::type;
01146
01151     template<typename v1, typename v2>
01152     using sub_t = typename sub<v1, v2>::type;
01153
01158     template<typename v1, typename v2>
01159     using mul_t = typename mul<v1, v2>::type;
01160
01165     template<typename v1, typename v2>
01166     using div_t = typename div<v1, v2>::type;
01167
01172     template<typename v1, typename v2>
01173     using mod_t = typename remainder<v1, v2>::type;
01174
01179     template<typename v1, typename v2>

```



```

01180     using gt_t = typename gt<v1, v2>::type;
01181
01186     template<typename v1, typename v2>
01187     using lt_t = typename lt<v1, v2>::type;
01188
01193     template<typename v1, typename v2>
01194     using eq_t = typename eq<v1, v2>::type;
01195
01199     template<typename v1, typename v2>
01200     static constexpr bool eq_v = eq_t<v1, v2>::value;
01201
01206     template<typename v1, typename v2>
01207     using gcd_t = gcd_t<i32, v1, v2>;
01208
01212     template<typename v>
01213     using pos_t = typename pos<v>::type;
01214
01218     template<typename v>
01219     static constexpr bool pos_v = pos_t<v>::value;
01220 };
01221 } // namespace aerobus
01222
01223 // i64
01224 namespace aerobus {
01226     struct i64 {
01228         using inner_type = int64_t;
01231         template<int64_t x>
01232         struct val {
01234             using inner_type = int32_t;
01236             using enclosing_type = i64;
01238             static constexpr int64_t v = x;
01239
01242             template<typename valueType>
01243             static constexpr INLINED_DEVICE valueType get() {
01244                 return static_cast<valueType>(x);
01245             }
01246
01248             using is_zero_t = std::bool_constant<x == 0>;
01249
01251             static std::string to_string() {
01252                 return std::to_string(x);
01253             }
01254         };
01255
01258         template<auto x>
01259         using inject_constant_t = val<static_cast<int64_t>(x)>;
01260
01265         template<typename v>
01266         using inject_ring_t = v;
01267
01269         using zero = val<0>;
01271         using one = val<1>;
01273         static constexpr bool is_field = false;
01275         static constexpr bool is_euclidean_domain = true;
01276
01277     private:
01278         template<typename v1, typename v2>
01279         struct add {
01280             using type = val<v1::v + v2::v>;
01281         };
01282
01283         template<typename v1, typename v2>
01284         struct sub {
01285             using type = val<v1::v - v2::v>;
01286         };
01287
01288         template<typename v1, typename v2>
01289         struct mul {
01290             using type = val<v1::v * v2::v>;
01291         };
01292
01293         template<typename v1, typename v2>
01294         struct div {
01295             using type = val<v1::v / v2::v>;
01296         };
01297
01298         template<typename v1, typename v2>
01299         struct remainder {
01300             using type = val<v1::v % v2::v>;
01301         };
01302
01303         template<typename v1, typename v2>
01304         struct gt {
01305             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01306         };
01307
01308         template<typename v1, typename v2>

```

```

01309     struct lt {
01310         using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01311     };
01312
01313     template<typename v1, typename v2>
01314     struct eq {
01315         using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01316     };
01317
01318     template<typename v>
01319     struct pos {
01320         using type = std::bool_constant<(v::v > 0)>;
01321     };
01322
01323 public:
01324     template<typename v1, typename v2>
01325     using add_t = typename add<v1, v2>::type;
01326
01327     template<typename v1, typename v2>
01328     using sub_t = typename sub<v1, v2>::type;
01329
01330     template<typename v1, typename v2>
01331     using mul_t = typename mul<v1, v2>::type;
01332
01333     template<typename v1, typename v2>
01334     using div_t = typename div<v1, v2>::type;
01335
01336     template<typename v1, typename v2>
01337     using mod_t = typename remainder<v1, v2>::type;
01338
01339     template<typename v1, typename v2>
01340     using gt_t = typename gt<v1, v2>::type;
01341
01342     template<typename v1, typename v2>
01343     static constexpr bool gt_v = gt_t<v1, v2>::value;
01344
01345     template<typename v1, typename v2>
01346     using lt_t = typename lt<v1, v2>::type;
01347
01348     template<typename v1, typename v2>
01349     static constexpr bool lt_v = lt_t<v1, v2>::value;
01350
01351     template<typename v1, typename v2>
01352     using eq_t = typename eq<v1, v2>::type;
01353
01354     template<typename v1, typename v2>
01355     static constexpr bool eq_v = eq_t<v1, v2>::value;
01356
01357     template<typename v1, typename v2>
01358     using gcd_t = gcd_t<i64, v1, v2>;
01359
01360     template<typename v>
01361     using pos_t = typename pos<v>::type;
01362
01363     template<typename v>
01364     static constexpr bool pos_v = pos_t<v>::value;
01365 };
01366
01367 template<>
01368 struct Embed<i32, i64> {
01369     template<typename val>
01370     using type = i64::val<static_cast<int64_t>(val::v)>;
01371 };
01372 } // namespace aerobus
01373
01374 // z/pz
01375 namespace aerobus {
01376     template<int32_t p>
01377     struct zpz {
01378         using inner_type = int32_t;
01379
01380         template<int32_t x>
01381         struct val {
01382             using enclosing_type = zpz<p>;
01383             static constexpr int32_t v = x % p;
01384
01385             template<typename valueType>
01386             static constexpr INLINED_DEVICE valueType get() {
01387                 return static_cast<valueType>(x % p);
01388             }
01389
01390             using is_zero_t = std::bool_constant<v == 0>;
01391
01392             static constexpr bool is_zero_v = v == 0;
01393
01394             static std::string to_string() {
01395                 return std::to_string(x % p);
01396             }
01397         };
01398     };
01399 }

```

```

01465     }
01466 };
01467
01470     template<auto x>
01471     using inject_constant_t = val<static_cast<int32_t>(x)>;
01472
01474     using zero = val<0>;
01475
01477     using one = val<1>;
01478
01480     static constexpr bool is_prime<p>::value;
01481
01483     static constexpr bool is_euclidean_domain = true;
01484
01485 private:
01486     template<typename v1, typename v2>
01487     struct add {
01488         using type = val<(v1::v + v2::v) % p>;
01489     };
01490
01491     template<typename v1, typename v2>
01492     struct sub {
01493         using type = val<(v1::v - v2::v) % p>;
01494     };
01495
01496     template<typename v1, typename v2>
01497     struct mul {
01498         using type = val<(v1::v * v2::v) % p>;
01499     };
01500
01501     template<typename v1, typename v2>
01502     struct div {
01503         using type = val<(v1::v % p) / (v2::v % p)>;
01504     };
01505
01506     template<typename v1, typename v2>
01507     struct remainder {
01508         using type = val<(v1::v % v2::v) % p>;
01509     };
01510
01511     template<typename v1, typename v2>
01512     struct gt {
01513         using type = std::conditional_t<(v1::v % p > v2::v % p), std::true_type, std::false_type>;
01514     };
01515
01516     template<typename v1, typename v2>
01517     struct lt {
01518         using type = std::conditional_t<(v1::v % p < v2::v % p), std::true_type, std::false_type>;
01519     };
01520
01521     template<typename v1, typename v2>
01522     struct eq {
01523         using type = std::conditional_t<(v1::v % p == v2::v % p), std::true_type, std::false_type>;
01524     };
01525
01526     template<typename v1>
01527     struct pos {
01528         using type = std::bool_constant<(v1::v > 0)>;
01529     };
01530
01531 public:
01532     template<typename v1, typename v2>
01533     using add_t = typename add<v1, v2>::type;
01534
01537     template<typename v1, typename v2>
01538     using sub_t = typename sub<v1, v2>::type;
01539
01541     template<typename v1, typename v2>
01542     using mul_t = typename mul<v1, v2>::type;
01543
01547     template<typename v1, typename v2>
01548     using div_t = typename div<v1, v2>::type;
01549
01553     template<typename v1, typename v2>
01554     using mod_t = typename remainder<v1, v2>::type;
01555
01559     template<typename v1, typename v2>
01560     using gt_t = typename gt<v1, v2>::type;
01561
01565     template<typename v1, typename v2>
01566     using gt_v = gt_t<v1, v2>::value;
01567
01571     template<typename v1, typename v2>
01572     using lt_t = typename lt<v1, v2>::type;
01573
01577     template<typename v1, typename v2>
01578     using lt_v = lt_t<v1, v2>::value;
01579
01583     template<typename v1, typename v2>
01584     static constexpr bool lt_v = lt_t<v1, v2>::value;

```

```

01585
01589     template<typename v1, typename v2>
01590     using eq_t = typename eq<v1, v2>::type;
01591
01595     template<typename v1, typename v2>
01596     static constexpr bool eq_v = eq_t<v1, v2>::value;
01597
01601     template<typename v1, typename v2>
01602     using gcd_t = gcd_t<i32, v1, v2>;
01603
01606     template<typename v1>
01607     using pos_t = typename pos<v1>::type;
01608
01611     template<typename v>
01612     static constexpr bool pos_v = pos_t<v>::value;
01613 };
01614
01617 template<int32_t x>
01618 struct Embed<zpz<x>, i32> {
01621     template <typename val>
01622     using type = i32::val<val::v>;
01623 };
01624 } // namespace aerobus
01625
01626 // polynomial
01627 namespace aerobus {
01628     // coeffN x^N + ...
01633     template<typename Ring>
01634     requires IsEuclideanDomain<Ring>
01635     struct polynomial {
01636         static constexpr bool is_field = false;
01637         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01638
01641         template<typename P>
01642         struct horner_reduction_t {
01643             template<size_t index, size_t stop>
01644             struct inner {
01645                 template<typename accum, typename x>
01646                 using type = typename horner_reduction_t<P>::template inner<index + 1, stop>
01647                 ::template type<
01648                     typename Ring::template add_t<
01649                         typename Ring::template mul_t<x, accum>,
01650                         typename P::template coeff_at_t<P::degree - index>
01651                     >, x>;
01652             };
01653
01654             template<size_t stop>
01655             struct inner<stop, stop> {
01656                 template<typename accum, typename x>
01657                 using type = accum;
01658             };
01659         };
01660
01664         template<typename coeffN, typename... coeffs>
01665         struct val {
01667             using ring_type = Ring;
01669             using enclosing_type = polynomial<Ring>;
01671             static constexpr size_t degree = sizeof...(coeffs);
01673             using aN = coeffN;
01675             using strip = val<coeffs...>;
01677             using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01679             static constexpr bool is_zero_v = is_zero_t::value;
01680
01681         private:
01682             template<size_t index, typename E = void>
01683             struct coeff_at {};
01684
01685             template<size_t index>
01686             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))> {
01687                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01688             };
01689
01690             template<size_t index>
01691             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))> {
01692                 using type = typename Ring::zero;
01693             };
01694
01695         public:
01698             template<size_t index>
01699             using coeff_at_t = typename coeff_at<index>::type;
01700
01703             static std::string to_string() {
01704                 return string_helper<coeffN, coeffs...>::func();
01705             }
01706
01711             template<typename arithmeticType>
01712             static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {

```

```

01713         #ifdef WITH_CUDA_FP16
01714         arithmeticType start;
01715         if constexpr (std::is_same_v<arithmeticType, __half2>) {
01716             start = __half2(0, 0);
01717         } else {
01718             start = static_cast<arithmeticType>(0);
01719         }
01720         #else
01721         arithmeticType start = static_cast<arithmeticType>(0);
01722         #endif
01723         return horner_evaluation<arithmeticType, val>
01724             ::template inner<0, degree + 1>
01725             ::func(start, x);
01726     }
01727
01740     template<typename arithmeticType>
01741     static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01742         return compensated_horner<arithmeticType, val>::func(x);
01743     }
01744
01745     template<typename x>
01746     using value_at_t = horner_reduction_t<val>
01747         ::template inner<0, degree + 1>
01748         ::template type<typename Ring::zero, x>;
01749 };
01750
01753     template<typename coeffN>
01754     struct val<coeffN> {
01755         using ring_type = Ring;
01756         using enclosing_type = polynomial<Ring>;
01757         static constexpr size_t degree = 0;
01758         using aN = coeffN;
01759         using strip = val<coeffN>;
01760         using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01761
01762         static constexpr bool is_zero_v = is_zero_t::value;
01763
01764         template<size_t index, typename E = void>
01765         struct coeff_at {};
01766
01767         template<size_t index>
01768         struct coeff_at<index, std::enable_if_t<(index == 0)>> {
01769             using type = aN;
01770         };
01771
01772         template<size_t index>
01773         struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)>> {
01774             using type = typename Ring::zero;
01775         };
01776
01777         template<size_t index>
01778         using coeff_at_t = typename coeff_at<index>::type;
01779
01780         static std::string to_string() {
01781             return string_helper<coeffN>::func();
01782         }
01783
01784         template<typename arithmeticType>
01785         static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01786             return coeffN::template get<arithmeticType>();
01787         }
01788
01789         template<typename arithmeticType>
01790         static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01791             return coeffN::template get<arithmeticType>();
01792         }
01793
01794         template<typename x>
01795         using value_at_t = coeffN;
01796     };
01797
01798     using zero = val<typename Ring::zero>;
01799     using one = val<typename Ring::one>;
01800     using X = val<typename Ring::one, typename Ring::zero>;
01801
01802 private:
01803     template<typename P, typename E = void>
01804     struct simplify;
01805
01806     template<typename P1, typename P2, typename I>
01807     struct add_low;
01808
01809     template<typename P1, typename P2>
01810     struct add {
01811         using type = typename simplify<typename add_low<
01812             P1,
01813             P2,

```

```

01820         internal::make_index_sequence_reverse<
01821         std::max(P1::degree, P2::degree) + 1
01822         >::type>::type;
01823     };
01824
01825     template <typename P1, typename P2, typename I>
01826     struct sub_low;
01827
01828     template <typename P1, typename P2, typename I>
01829     struct mul_low;
01830
01831     template<typename v1, typename v2>
01832     struct mul {
01833         using type = typename mul_low<
01834             v1,
01835             v2,
01836             internal::make_index_sequence_reverse<
01837             v1::degree + v2::degree + 1
01838             >::type;
01839         };
01840
01841     template<typename coeff, size_t deg>
01842     struct monomial;
01843
01844     template<typename v, typename E = void>
01845     struct derive_helper {};
01846
01847     template<typename v>
01848     struct derive_helper<v, std::enable_if_t<v::degree == 0> {
01849         using type = zero;
01850     };
01851
01852     template<typename v>
01853     struct derive_helper<v, std::enable_if_t<v::degree != 0> {
01854         using type = typename add<
01855             typename derive_helper<typename simplify<typename v::strip>::type>::type,
01856             typename monomial<
01857                 typename Ring::template mul_t<
01858                     typename v::aN,
01859                     typename Ring::template inject_constant_t<(v::degree)>
01860                 >,
01861                 v::degree - 1
01862             >::type
01863             >::type;
01864     };
01865
01866     template<typename v1, typename v2, typename E = void>
01867     struct eq_helper {};
01868
01869     template<typename v1, typename v2>
01870     struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree> {
01871         using type = std::false_type;
01872     };
01873
01874
01875     template<typename v1, typename v2>
01876     struct eq_helper<v1, v2, std::enable_if_t<
01877         v1::degree == v2::degree &&
01878         (v1::degree != 0 || v2::degree != 0) &&
01879         std::is_same<
01880             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01881             std::false_type
01882         >::value
01883         > {
01884     > {
01885         using type = std::false_type;
01886     };
01887
01888     template<typename v1, typename v2>
01889     struct eq_helper<v1, v2, std::enable_if_t<
01890         v1::degree == v2::degree &&
01891         (v1::degree != 0 || v2::degree != 0) &&
01892         std::is_same<
01893             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01894             std::true_type
01895         >::value
01896         > {
01897         using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01898     };
01899
01900     template<typename v1, typename v2>
01901     struct eq_helper<v1, v2, std::enable_if_t<
01902         v1::degree == v2::degree &&
01903         (v1::degree == 0)
01904         > {
01905         using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01906     };

```

```

01907
01908     template<typename v1, typename v2, typename E = void>
01909     struct lt_helper {};
01910
01911     template<typename v1, typename v2>
01912     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)>> {
01913         using type = std::true_type;
01914     };
01915
01916     template<typename v1, typename v2>
01917     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)>> {
01918         using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01919     };
01920
01921     template<typename v1, typename v2>
01922     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)>> {
01923         using type = std::false_type;
01924     };
01925
01926     template<typename v1, typename v2, typename E = void>
01927     struct gt_helper {};
01928
01929     template<typename v1, typename v2>
01930     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)>> {
01931         using type = std::true_type;
01932     };
01933
01934     template<typename v1, typename v2>
01935     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)>> {
01936         using type = std::false_type;
01937     };
01938
01939     template<typename v1, typename v2>
01940     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)>> {
01941         using type = std::false_type;
01942     };
01943
01944     // when high power is zero : strip
01945     template<typename P>
01946     struct simplify<P, std::enable_if_t<
01947         std::is_same<
01948             typename Ring::zero,
01949             typename P::aN
01950         >::value && (P::degree > 0)
01951     > {
01952         using type = typename simplify<typename P::strip>::type;
01953     };
01954
01955     // otherwise : do nothing
01956     template<typename P>
01957     struct simplify<P, std::enable_if_t<
01958         !std::is_same<
01959             typename Ring::zero,
01960             typename P::aN
01961         >::value && (P::degree > 0)
01962     > {
01963         using type = P;
01964     };
01965
01966     // do not simplify constants
01967     template<typename P>
01968     struct simplify<P, std::enable_if_t<P::degree == 0>> {
01969         using type = P;
01970     };
01971
01972     // addition at
01973     template<typename P1, typename P2, size_t index>
01974     struct add_at {
01975         using type =
01976             typename Ring::template add_t<
01977                 typename P1::template coeff_at_t<index>,
01978                 typename P2::template coeff_at_t<index>>;
01979     };
01980
01981     template<typename P1, typename P2, size_t index>
01982     using add_at_t = typename add_at<P1, P2, index>::type;
01983
01984     template<typename P1, typename P2, std::size_t... I>
01985     struct add_low<P1, P2, std::index_sequence<I...>> {
01986         using type = val<add_at_t<P1, P2, I>...>;
01987     };
01988
01989     // subtraction at
01990     template<typename P1, typename P2, size_t index>
01991     struct sub_at {
01992         using type =
01993             typename Ring::template sub_t<

```

```

01994         typename P1::template coeff_at_t<index>,
01995         typename P2::template coeff_at_t<index>;
01996     };
01997
01998     template<typename P1, typename P2, size_t index>
01999     using sub_at_t = typename sub_at<P1, P2, index>::type;
02000
02001     template<typename P1, typename P2, std::size_t... I>
02002     struct sub_low<P1, P2, std::index_sequence<I...> {
02003         using type = val<sub_at_t<P1, P2, I>...>;
02004     };
02005
02006     template<typename P1, typename P2>
02007     struct sub {
02008         using type = typename simplify<typename sub_low<
02009             P1,
02010             P2,
02011             internal::make_index_sequence_reverse<
02012                 std::max(P1::degree, P2::degree) + 1
02013             >::type>::type;
02014     };
02015
02016     // multiplication at
02017     template<typename v1, typename v2, size_t k, size_t index, size_t stop>
02018     struct mul_at_loop_helper {
02019         using type = typename Ring::template add_t<
02020             typename Ring::template mul_t<
02021                 typename v1::template coeff_at_t<index>,
02022                 typename v2::template coeff_at_t<k - index>
02023             >,
02024             typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
02025         >;
02026     };
02027
02028     template<typename v1, typename v2, size_t k, size_t stop>
02029     struct mul_at_loop_helper<v1, v2, k, stop, stop> {
02030         using type = typename Ring::template mul_t<
02031             typename v1::template coeff_at_t<stop>,
02032             typename v2::template coeff_at_t<0>;
02033     };
02034
02035     template <typename v1, typename v2, size_t k, typename E = void>
02036     struct mul_at {};
02037
02038     template<typename v1, typename v2, size_t k>
02039     struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)> {
02040         using type = typename Ring::zero;
02041     };
02042
02043     template<typename v1, typename v2, size_t k>
02044     struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)> {
02045         using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
02046     };
02047
02048     template<typename P1, typename P2, size_t index>
02049     using mul_at_t = typename mul_at<P1, P2, index>::type;
02050
02051     template<typename P1, typename P2, std::size_t... I>
02052     struct mul_low<P1, P2, std::index_sequence<I...> {
02053         using type = val<mul_at_t<P1, P2, I>...>;
02054     };
02055
02056     // division helper
02057     template< typename A, typename B, typename Q, typename R, typename E = void>
02058     struct div_helper {};
02059
02060     template<typename A, typename B, typename Q, typename R>
02061     struct div_helper<A, B, Q, R, std::enable_if_t<
02062         (R::degree < B::degree) ||
02063         (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)> {
02064         using q_type = Q;
02065         using mod_type = R;
02066         using gcd_type = B;
02067     };
02068
02069     template<typename A, typename B, typename Q, typename R>
02070     struct div_helper<A, B, Q, R, std::enable_if_t<
02071         (R::degree >= B::degree) &&
02072         !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)> {
02073     private: // NOLINT
02074         using rN = typename R::aN;
02075         using bN = typename B::aN;
02076         using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
02077             B::degree>::type;
02078         using rr = typename sub<R, typename mul<pT, B>::type>::type;
02079         using qq = typename add<Q, pT>::type;

```



```

02080     public:
02081         using q_type = typename div_helper<A, B, qq, rr>::q_type;
02082         using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
02083         using gcd_type = rr;
02084     };
02085
02086     template<typename A, typename B>
02087     struct div {
02088         static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
02089         using q_type = typename div_helper<A, B, zero, A>::q_type;
02090         using m_type = typename div_helper<A, B, zero, A>::mod_type;
02091     };
02092
02093     template<typename P>
02094     struct make_unit {
02095         using type = typename div<P, val<typename P::aN>::q_type>;
02096     };
02097
02098     template<typename coeff, size_t deg>
02099     struct monomial {
02100         using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
02101     };
02102
02103     template<typename coeff>
02104     struct monomial<coeff, 0> {
02105         using type = val<coeff>;
02106     };
02107
02108     template<typename arithmeticType, typename P>
02109     struct horner_evaluation {
02110         template<size_t index, size_t stop>
02111         struct inner {
02112             static constexpr DEVICE INLINED arithmeticType func(
02113                 const arithmeticType& accum, const arithmeticType& x) {
02114                 return horner_evaluation<arithmeticType, P>::template inner<index + 1,
stop>::func(
02115                     internal::fma_helper<arithmeticType>::eval(
02116                         x,
02117                         accum,
02118                         P::template coeff_at_t<P::degree - index>::template
get<arithmeticType>(), x);
02119                     );
02120             };
02121
02122             template<size_t stop>
02123             struct inner<stop, stop> {
02124                 static constexpr DEVICE INLINED arithmeticType func(
02125                     const arithmeticType& accum, const arithmeticType& x) {
02126                     return accum;
02127                 }
02128             };
02129         };
02130
02131         template<typename arithmeticType, typename P>
02132         struct compensated_horner {
02133             template<int64_t index, int ghost>
02134             struct EFTHorner {
02135                 static INLINED DEVICE void func(
02136                     arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType
*r) {
02137                     arithmeticType p;
02138                     internal::two_prod(*r, x, &p, pi + P::degree - index - 1);
02139                     constexpr arithmeticType coeff = P::template coeff_at_t<index>::template
get<arithmeticType>();
02140                     internal::two_sum<arithmeticType>(
02141                         p, coeff,
02142                         r, sigma + P::degree - index - 1);
02143                     EFTHorner<index - 1, ghost>::func(x, pi, sigma, r);
02144                 }
02145             };
02146
02147             template<int ghost>
02148             struct EFTHorner<-1, ghost> {
02149                 static INLINED DEVICE void func(
02150                     arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType
*r) {
02151                 }
02152             };
02153
02154             static INLINED DEVICE arithmeticType func(arithmeticType x) {
02155                 arithmeticType pi[P::degree], sigma[P::degree];
02156                 arithmeticType r = P::template coeff_at_t<P::degree>::template get<arithmeticType>();
02157                 EFTHorner<P::degree - 1, 0>::func(x, pi, sigma, &r);
02158                 arithmeticType c = internal::horner<arithmeticType, P::degree - 1>(pi, sigma, x);
02159                 return r + c;
02160             }
02161         };

```

```

02162
02163     template<typename coeff, typename... coeffs>
02164     struct string_helper {
02165         static std::string func() {
02166             std::string tail = string_helper<coeffs...>::func();
02167             std::string result = "";
02168             if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
02169                 return tail;
02170             } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
02171                 if (sizeof...(coeffs) == 1) {
02172                     result += "x";
02173                 } else {
02174                     result += "x^" + std::to_string(sizeof...(coeffs));
02175                 }
02176             } else {
02177                 if (sizeof...(coeffs) == 1) {
02178                     result += coeff::to_string() + " x";
02179                 } else {
02180                     result += coeff::to_string()
02181                         + " x^" + std::to_string(sizeof...(coeffs));
02182                 }
02183             }
02184
02185             if (!tail.empty()) {
02186                 if (tail.at(0) != '-' ) {
02187                     result += " + " + tail;
02188                 } else {
02189                     result += " - " + tail.substr(1);
02190                 }
02191             }
02192
02193             return result;
02194         }
02195     };
02196
02197     template<typename coeff>
02198     struct string_helper<coeff> {
02199         static std::string func() {
02200             if (!std::is_same<coeff, typename Ring::zero>::value) {
02201                 return coeff::to_string();
02202             } else {
02203                 return "";
02204             }
02205         }
02206     };
02207
02208 public:
02209     template<typename P>
02210     using simplify_t = typename simplify<P>::type;
02211
02212     template<typename v1, typename v2>
02213     using add_t = typename add<v1, v2>::type;
02214
02215     template<typename v1, typename v2>
02216     using sub_t = typename sub<v1, v2>::type;
02217
02218     template<typename v1, typename v2>
02219     using mul_t = typename mul<v1, v2>::type;
02220
02221     template<typename v1, typename v2>
02222     using eq_t = typename eq_helper<v1, v2>::type;
02223
02224     template<typename v1, typename v2>
02225     using lt_t = typename lt_helper<v1, v2>::type;
02226
02227     template<typename v1, typename v2>
02228     using gt_t = typename gt_helper<v1, v2>::type;
02229
02230     template<typename v1, typename v2>
02231     using div_t = typename div<v1, v2>::q_type;
02232
02233     template<typename v1, typename v2>
02234     using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
02235
02236     template<typename coeff, size_t deg>
02237     using monomial_t = typename monomial<coeff, deg>::type;
02238
02239     template<typename v>
02240     using derive_t = typename derive_helper<v>::type;
02241
02242     template<typename v>
02243     using pos_t = typename Ring::template pos_t<typename v::aN>;
02244
02245     template<typename v>
02246     static constexpr bool pos_v = pos_t<v>::value;
02247
02248     template<typename v1, typename v2>

```

```

02287     using gcd_t = std::conditional_t<
02288         Ring::is_euclidean_domain,
02289         typename make_unit<gcd_t<polynomial<Ring>, v1, v2>::type,
02290         void>;
02291
02292     template<auto x>
02293     using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
02294
02295     template<typename v>
02296     using inject_ring_t = val<v>;
02297
02298 };
02299 // namespace aerobus
02300
02301 // fraction field
02302 namespace aerobus {
02303     namespace internal {
02304         template<typename Ring, typename E = void>
02305         requires IsEuclideanDomain<Ring>
02306         struct _FractionField {};
02307
02308         template<typename Ring>
02309         requires IsEuclideanDomain<Ring>
02310         struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain>> {
02311             static constexpr bool is_field = true;
02312             static constexpr bool is_euclidean_domain = true;
02313
02314         private:
02315             template<typename val1, typename val2, typename E = void>
02316             struct to_string_helper {};
02317
02318             template<typename val1, typename val2>
02319             struct to_string_helper<val1, val2,
02320                 std::enable_if_t<
02321                     Ring::template eq_t<
02322                         val2, typename Ring::one
02323                         >::value
02324                     >
02325                 > {
02326                 static std::string func() {
02327                     return val1::to_string();
02328                 }
02329             };
02330
02331             template<typename val1, typename val2>
02332             struct to_string_helper<val1, val2,
02333                 std::enable_if_t<
02334                     !Ring::template eq_t<
02335                         val2,
02336                         typename Ring::one
02337                         >::value
02338                     >
02339                 > {
02340                 static std::string func() {
02341                     return "(" + val1::to_string() + " ) / ( " + val2::to_string() + " )";
02342                 }
02343             };
02344
02345         public:
02346             template<typename val1, typename val2>
02347             struct val {
02348                 using x = val1;
02349                 using y = val2;
02350                 using is_zero_t = typename val1::is_zero_t;
02351                 static constexpr bool is_zero_v = val1::is_zero_t::value;
02352
02353                 using ring_type = Ring;
02354                 using enclosing_type = _FractionField<Ring>;
02355
02356                 static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
02357
02358                 template<typename valueType, int ghost = 0>
02359                 struct get_helper {
02360                     static constexpr INLINED_DEVICE valueType get() {
02361                         return internal::staticcast<valueType, typename
02362 ring_type::inner_type>::template func<x::v>() /
02363 internal::staticcast<valueType, typename ring_type::inner_type>::template
02364 func<y::v>();
02365                     }
02366                 };
02367
02368                 #ifdef WITH_CUDA_FP16
02369                 template<int ghost>
02370                 struct get_helper<__half, ghost> {
02371                     static constexpr INLINED_DEVICE __half get() {
02372                         return internal::my_float2half_rn(
02373 internal::staticcast<float, typename ring_type::inner_type>::template
02374 func<x::v>() /

```

```

02386             internal::staticcast<float, typename ring_type::inner_type>::template
func<y::v>());
02387     }
02388 };
02389
02390     template<int ghost>
02391     struct get_helper<__half2, ghost> {
02392         static constexpr INLINED DEVICE __half2 get() {
02393             constexpr __half tmp = internal::my_float2half_rn(
02394                 internal::staticcast<float, typename ring_type::inner_type>::template
02395 func<x::v>() /
02396             internal::staticcast<float, typename ring_type::inner_type>::template
02397 func<y::v>());
02398         return __half2(tmp, tmp);
02399     }
02400 };
02401 #endif
02402
02403     template<typename valueType>
02404     static constexpr INLINED DEVICE valueType get() {
02405         return get_helper<valueType, 0>::get();
02406     }
02407
02408     static std::string to_string() {
02409         return to_string_helper<val1, val2>::func();
02410     }
02411
02412     template<typename arithmeticType>
02413     static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& v) {
02414         return x::eval(v) / y::eval(v);
02415     }
02416 };
02417
02418 using zero = val<typename Ring::zero, typename Ring::one>;
02419 using one = val<typename Ring::one, typename Ring::one>;
02420
02421     template<typename v>
02422     using inject_t = val<v, typename Ring::one>;
02423
02424     template<auto x>
02425     using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
02426 Ring::one>;
02427
02428     template<typename v>
02429     using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
02430
02431     using ring_type = Ring;
02432
02433 private:
02434     template<typename v, typename E = void>
02435     struct simplify {};
02436
02437     // x = 0
02438     template<typename v>
02439     struct simplify<v, std::enable_if_t<v::x::is_zero_t::value> {
02440         using type = typename _FractionField<Ring>::zero;
02441     };
02442
02443     // x != 0
02444     template<typename v>
02445     struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value> {
02446     private:
02447         using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
02448         using newx = typename Ring::template div_t<typename v::x, _gcd>;
02449         using newy = typename Ring::template div_t<typename v::y, _gcd>;
02450
02451         using posx = std::conditional_t<
02452             !Ring::template pos_v<newy>,
02453             typename Ring::template sub_t<typename Ring::zero, newx>,
02454             newx>;
02455         using posy = std::conditional_t<
02456             !Ring::template pos_v<newy>,
02457             typename Ring::template sub_t<typename Ring::zero, newy>,
02458             newy>;
02459     public:
02460         using type = typename _FractionField<Ring>::template val<posx, posy>;
02461     };
02462
02463     public:
02464         template<typename v>
02465         using simplify_t = typename simplify<v>::type;
02466
02467     private:
02468         template<typename v1, typename v2>
02469         struct add {
02470         private:
02471             using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;

```

```

02489         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02490         using dividend = typename Ring::template add_t<a, b>;
02491         using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02492         using g = typename Ring::template gcd_t<dividend, diviser>;
02493
02494     public:
02495         using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
diviser>>;
02496     };
02497
02498     template<typename v>
02499     struct pos {
02500         using type = std::conditional_t<
02501             (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
02502             (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
02503             std::true_type,
02504             std::false_type>;
02505     };
02506
02507     template<typename v1, typename v2>
02508     struct sub {
02509     private:
02510         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02511         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02512         using dividend = typename Ring::template sub_t<a, b>;
02513         using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02514         using g = typename Ring::template gcd_t<dividend, diviser>;
02515
02516     public:
02517         using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
diviser>>;
02518     };
02519
02520     template<typename v1, typename v2>
02521     struct mul {
02522     private:
02523         using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
02524         using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02525
02526     public:
02527         using type = typename _FractionField<Ring>::template simplify_t<val<a, b>>;
02528     };
02529
02530     template<typename v1, typename v2, typename E = void>
02531     struct div {};
02532
02533     template<typename v1, typename v2>
02534     struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
_FractionField<Ring>::zero::value>> {
02535     private:
02536         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02537         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02538
02539     public:
02540         using type = typename _FractionField<Ring>::template simplify_t<val<a, b>>;
02541     };
02542
02543     template<typename v1, typename v2>
02544     struct div<v1, v2, std::enable_if_t<
std::is_same<zero, v1>::value && std::is_same<v2, zero>::value>> {
02545         using type = one;
02546     };
02547
02548     template<typename v1, typename v2>
02549     struct eq {
02550     private:
02551         using type = std::conditional_t<
02552             std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
02553             std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
02554             std::true_type,
02555             std::false_type>;
02556     };
02557
02558     template<typename v1, typename v2, typename E = void>
02559     struct gt;
02560
02561     template<typename v1, typename v2>
02562     struct gt<v1, v2, std::enable_if_t<
(eq<v1, v2>::type::value)
02563         >> {
02564         using type = std::false_type;
02565     };
02566
02567     template<typename v1, typename v2>
02568     struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02569         (!pos<v1>::type::value) && (!pos<v2>::type::value)
02570         >> {
02571

```

```

02573         using type = typename gt<
02574             typename sub<zero, v1>::type, typename sub<zero, v2>::type
02575         >::type;
02576     };
02577
02578     template<typename v1, typename v2>
02579     struct gt<v1, v2, std::enable_if_t<
02580         (!eq<v1, v2>::type::value) &&
02581         (pos<v1>::type::value) && (!pos<v2>::type::value)
02582     >> {
02583         using type = std::true_type;
02584     };
02585
02586     template<typename v1, typename v2>
02587     struct gt<v1, v2, std::enable_if_t<
02588         (!eq<v1, v2>::type::value) &&
02589         (!pos<v1>::type::value) && (pos<v2>::type::value)
02590     >> {
02591         using type = std::false_type;
02592     };
02593
02594     template<typename v1, typename v2>
02595     struct gt<v1, v2, std::enable_if_t<
02596         (!eq<v1, v2>::type::value) &&
02597         (pos<v1>::type::value) && (pos<v2>::type::value)
02598     >> {
02599         using type = typename Ring::template gt_t<
02600             typename Ring::template mul_t<v1::x, v2::y>,
02601             typename Ring::template mul_t<v2::y, v2::x>
02602         >;
02603     };
02604
02605 public:
02606     template<typename v1, typename v2>
02607     using add_t = typename add<v1, v2>::type;
02608
02609     template<typename v1, typename v2>
02610     using mod_t = zero;
02611
02612     template<typename v1, typename v2>
02613     using gcd_t = v1;
02614
02615     template<typename v1, typename v2>
02616     using sub_t = typename sub<v1, v2>::type;
02617
02618     template<typename v1, typename v2>
02619     using mul_t = typename mul<v1, v2>::type;
02620
02621     template<typename v1, typename v2>
02622     using div_t = typename div<v1, v2>::type;
02623
02624     template<typename v1, typename v2>
02625     using eq_t = typename eq<v1, v2>::type;
02626
02627     template<typename v1, typename v2>
02628     static constexpr bool eq_v = eq<v1, v2>::type::value;
02629
02630     template<typename v1, typename v2>
02631     using gt_t = typename gt<v1, v2>::type;
02632
02633     template<typename v1, typename v2>
02634     static constexpr bool gt_v = gt<v1, v2>::type::value;
02635
02636     template<typename v1>
02637     using pos_t = typename pos<v1>::type;
02638
02639     template<typename v>
02640     static constexpr bool pos_v = pos_t<v>::value;
02641 };
02642
02643 template<typename Ring, typename E = void>
02644 requires IsEuclideanDomain<Ring>
02645 struct FractionFieldImpl {};
02646
02647 // fraction field of a field is the field itself
02648 template<typename Field>
02649 requires IsEuclideanDomain<Field>
02650 struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field> {
02651     using type = Field;
02652     template<typename v>
02653     using inject_t = v;
02654 };
02655
02656 // fraction field of a ring is the actual fraction field
02657 template<typename Ring>
02658 requires IsEuclideanDomain<Ring>
02659 struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field> {

```

```

02696         using type = _FractionField<Ring>;
02697     };
02698 } // namespace internal
02699
02700 template<typename Ring>
02701 requires IsEuclideanDomain<Ring>
02702 using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02703
02704 template<typename Ring>
02705 struct Embed<Ring, FractionField<Ring> > {
02706     template<typename v>
02707     using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02708 };
02709 } // namespace aerobus
02710
02711 // short names for common types
02712 namespace aerobus {
02713     template<typename X, typename Y>
02714     requires IsRing<typename X::enclosing_type> &&
02715     (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02716     using add_t = typename X::enclosing_type::template add_t<X, Y>;
02717
02718     template<typename X, typename Y>
02719     requires IsRing<typename X::enclosing_type> &&
02720     (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02721     using sub_t = typename X::enclosing_type::template sub_t<X, Y>;
02722
02723     template<typename X, typename Y>
02724     requires IsRing<typename X::enclosing_type> &&
02725     (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02726     using mul_t = typename X::enclosing_type::template mul_t<X, Y>;
02727
02728     template<typename X, typename Y>
02729     requires IsEuclideanDomain<typename X::enclosing_type> &&
02730     (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02731     using div_t = typename X::enclosing_type::template div_t<X, Y>;
02732
02733     using q32 = FractionField<i32>;
02734
02735     using fpq32 = FractionField<polynomial<q32>>;
02736
02737     using q64 = FractionField<i64>;
02738
02739     using pi64 = polynomial<i64>;
02740
02741     using pq64 = polynomial<q64>;
02742
02743     using fpq64 = FractionField<polynomial<q64>>;
02744
02745     template<typename Ring, typename v1, typename v2>
02746     using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02747
02748     template<typename v>
02749     using embed_int_poly_in_fractions_t =
02750         typename Embed<
02751             polynomial<typename v::ring_type>,
02752             polynomial<FractionField<typename v::ring_type>>::template type<v>;
02753
02754     template<int64_t p, int64_t q>
02755     using make_q64_t = typename q64::template simplify_t<
02756         typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>>;
02757
02758     template<int32_t p, int32_t q>
02759     using make_q32_t = typename q32::template simplify_t<
02760         typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>>;
02761
02762     template<typename Ring, typename v1, typename v2>
02763     using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02764     template<typename Ring, typename v1, typename v2>
02765     using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02766
02767     template<>
02768     struct Embed<q32, q64> {
02769         template<typename v>
02770         using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02771     };
02772
02773     template<typename Small, typename Large>
02774     struct Embed<polynomial<Small>, polynomial<Large>> {
02775     private:
02776         template<typename v, typename i>
02777         struct at_low;
02778
02779         template<typename v, size_t i>
02780         struct at_index {
02781             using type = typename Embed<Small, Large>::template

```

```

    type<typename v::template coeff_at_t<i>>;
02840     };
02841
02842     template<typename v, size_t... Is>
02843     struct at_low<v, std::index_sequence<Is...> {
02844         using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
02845     };
02846
02847     public:
02850     template<typename v>
02851     using type = typename at_low<v, typename internal::make_index_sequence_reverse<v::degree +
1>::type;
02852     };
02853
02857     template<typename Ring, auto... xs>
02858     using make_int_polynomial_t = typename polynomial<Ring>::template val<
02859         typename Ring::template inject_constant_t<xs>...>;
02860
02864     template<typename Ring, auto... xs>
02865     using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<
02866         typename FractionField<Ring>::template inject_constant_t<xs>...>;
02867 } // namespace aerobus
02868
02869 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02870 namespace aerobus {
02871     namespace internal {
02872         template<typename T, size_t x, typename E = void>
02873         struct factorial {};
02874
02875         template<typename T, size_t x>
02876         struct factorial<T, x, std::enable_if_t<(x > 0)> {
02877             private:
02878                 template<typename, size_t, typename>
02879                 friend struct factorial;
02880             public:
02881                 using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
x - 1>::type>;
02882                 static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02883             };
02884
02885         template<typename T>
02886         struct factorial<T, 0> {
02887             public:
02888                 using type = typename T::one;
02889                 static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02890             };
02891     } // namespace internal
02892
02896     template<typename T, size_t i>
02897     using factorial_t = typename internal::factorial<T, i>::type;
02898
02902     template<typename T, size_t i>
02903     inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02904
02905     namespace internal {
02906         template<typename T, size_t k, size_t n, typename E = void>
02907         struct combination_helper {};
02908
02909         template<typename T, size_t k, size_t n>
02910         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)> {
02911             using type = typename FractionField<T>::template mul_t<
02912                 typename combination_helper<T, k - 1, n - 1>::type,
02913                 makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02914             };
02915
02916         template<typename T, size_t k, size_t n>
02917         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)> {
02918             using type = typename combination_helper<T, n - k, n>::type;
02919             };
02920
02921         template<typename T, size_t n>
02922         struct combination_helper<T, 0, n> {
02923             using type = typename FractionField<T>::one;
02924             };
02925
02926         template<typename T, size_t k, size_t n>
02927         struct combination {
02928             using type = typename internal::combination_helper<T, k, n>::type::x;
02929             static constexpr typename T::inner_type value =
02930                 internal::combination_helper<T, k, n>::type::template get<typename
T::inner_type>();
02931             };
02932     } // namespace internal
02933
02936     template<typename T, size_t k, size_t n>

```



```

02937     using combination_t = typename internal::combination<T, k, n>::type;
02938
02943     template<typename T, size_t k, size_t n>
02944     inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02945
02946     namespace internal {
02947         template<typename T, size_t m>
02948         struct bernoulli;
02949
02950         template<typename T, typename accum, size_t k, size_t m>
02951         struct bernoulli_helper {
02952             using type = typename bernoulli_helper<
02953                 T,
02954                 addfractions_t<T,
02955                     accum,
02956                     mulfractions_t<T,
02957                         makefraction_t<T,
02958                             combination_t<T, k, m + 1>,
02959                             typename T::one>,
02960                             typename bernoulli<T, k>::type
02961                         >,
02962                 >,
02963                 k + 1,
02964                 m>::type;
02965         };
02966
02967         template<typename T, typename accum, size_t m>
02968         struct bernoulli_helper<T, accum, m, m> {
02969             using type = accum;
02970         };
02971
02972
02973
02974         template<typename T, size_t m>
02975         struct bernoulli {
02976             using type = typename FractionField<T>::template mul_t<
02977                 typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02978                 makefraction_t<T,
02979                     typename T::template val<static_cast<typename T::inner_type>(-1)>,
02980                     typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02981                 >
02982             >;
02983
02984             template<typename floatType>
02985             static constexpr floatType value = type::template get<floatType>();
02986         };
02987
02988         template<typename T>
02989         struct bernoulli<T, 0> {
02990             using type = typename FractionField<T>::one;
02991
02992             template<typename floatType>
02993             static constexpr floatType value = type::template get<floatType>();
02994         };
02995     } // namespace internal
02996
03000     template<typename T, size_t n>
03001     using bernoulli_t = typename internal::bernoulli<T, n>::type;
03002
03007     template<typename FloatType, typename T, size_t n>
03008     inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
03009
03010     // bell numbers
03011     namespace internal {
03012         template<typename T, size_t n, typename E = void>
03013         struct bell_helper;
03014
03015         template<typename T, size_t n>
03016         struct bell_helper<T, n, std::enable_if_t<(n > 1)>> {
03017             template<typename accum, size_t i, size_t stop>
03018             struct sum_helper {
03019             private:
03020                 using left = typename T::template mul_t<
03021                     combination_t<T, i, n-1>,
03022                     typename bell_helper<T, i>::type>;
03023                 using new_accum = typename T::template add_t<accum, left>;
03024             public:
03025                 using type = typename sum_helper<new_accum, i+1, stop>::type;
03026             };
03027
03028             template<typename accum, size_t stop>
03029             struct sum_helper<accum, stop, stop> {
03030                 using type = accum;
03031             };
03032
03033             using type = typename sum_helper<typename T::zero, 0, n>::type;
03034         };

```

```

03035
03036     template<typename T>
03037     struct bell_helper<T, 0> {
03038         using type = typename T::one;
03039     };
03040
03041     template<typename T>
03042     struct bell_helper<T, 1> {
03043         using type = typename T::one;
03044     };
03045 } // namespace internal
03046
03050 template<typename T, size_t n>
03051 using bell_t = typename internal::bell_helper<T, n>::type;
03052
03056 template<typename T, size_t n>
03057 static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
03058
03059 namespace internal {
03060     template<typename T, int k, typename E = void>
03061     struct alternate {};
03062
03063     template<typename T, int k>
03064     struct alternate<T, k, std::enable_if_t<k % 2 == 0> > {
03065         using type = typename T::one;
03066         static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
03067     };
03068
03069     template<typename T, int k>
03070     struct alternate<T, k, std::enable_if_t<k % 2 != 0> > {
03071         using type = typename T::template sub_t<typename T::zero, typename T::one>;
03072         static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
03073     };
03074 } // namespace internal
03075
03078 template<typename T, int k>
03079 using alternate_t = typename internal::alternate<T, k>::type;
03080
03083 template<typename T, size_t k>
03084 inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
03085
03086 namespace internal {
03087     template<typename T, int n, int k, typename E = void>
03088     struct stirling_l_helper {};
03089
03090     template<typename T>
03091     struct stirling_l_helper<T, 0, 0> {
03092         using type = typename T::one;
03093     };
03094
03095     template<typename T, int n>
03096     struct stirling_l_helper<T, n, 0, std::enable_if_t<(n > 0)> > {
03097         using type = typename T::zero;
03098     };
03099
03100     template<typename T, int n>
03101     struct stirling_l_helper<T, 0, n, std::enable_if_t<(n > 0)> > {
03102         using type = typename T::zero;
03103     };
03104
03105     template<typename T, int n, int k>
03106     struct stirling_l_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)> > {
03107         using type = typename T::template sub_t<
03108             typename stirling_l_helper<T, n-1, k-1>::type,
03109             typename T::template mul_t<
03110                 typename T::template inject_constant_t<n-1>,
03111                 typename stirling_l_helper<T, n-1, k>::type
03112             >;
03113     };
03114 } // namespace internal
03115
03120 template<typename T, int n, int k>
03121 using stirling_l_signed_t = typename internal::stirling_l_helper<T, n, k>::type;
03122
03127 template<typename T, int n, int k>
03128 using stirling_l_unsigned_t = abs_t<typename internal::stirling_l_helper<T, n, k>::type>;
03129
03134 template<typename T, int n, int k>
03135 static constexpr typename T::inner_type stirling_l_unsigned_v = stirling_l_unsigned_t<T, n, k>::v;
03136
03141 template<typename T, int n, int k>
03142 static constexpr typename T::inner_type stirling_l_signed_v = stirling_l_signed_t<T, n, k>::v;
03143
03144 namespace internal {
03145     template<typename T, int n, int k, typename E = void>

```

```

03146     struct stirling_2_helper {};
03147
03148     template<typename T, int n>
03149     struct stirling_2_helper<T, n, n, std::enable_if_t<(n >= 0)>> {
03150         using type = typename T::one;
03151     };
03152
03153     template<typename T, int n>
03154     struct stirling_2_helper<T, n, 0, std::enable_if_t<(n > 0)>> {
03155         using type = typename T::zero;
03156     };
03157
03158     template<typename T, int n>
03159     struct stirling_2_helper<T, 0, n, std::enable_if_t<(n > 0)>> {
03160         using type = typename T::zero;
03161     };
03162
03163     template<typename T, int n, int k>
03164     struct stirling_2_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0) && (k < n)>> {
03165         using type = typename T::template add_t<
03166             typename stirling_2_helper<T, n-1, k-1>::type,
03167             typename T::template mul_t<
03168                 typename T::template inject_constant_t<k>,
03169                 typename stirling_2_helper<T, n-1, k>::type
03170             >>;
03171     };
03172 } // namespace internal
03173
03174 template<typename T, int n, int k>
03175 using stirling_2_t = typename internal::stirling_2_helper<T, n, k>::type;
03176
03177 template<typename T, int n, int k>
03178 static constexpr typename T::inner_type stirling_2_v = stirling_2_t<T, n, k>::v;
03179
03180 namespace internal {
03181     template<typename T>
03182     struct pow_scalar {
03183         template<size_t p>
03184         static constexpr DEVICE INLINED T func(const T& x) { return p == 0 ? static_cast<T>(1) :
03185             p % 2 == 0 ? func<p/2>(x) * func<p/2>(x) :
03186             x * func<p/2>(x) * func<p/2>(x);
03187         }
03188     };
03189
03190     template<typename T, typename p, size_t n, typename E = void>
03191     requires IsEuclideanDomain<T>
03192     struct pow;
03193
03194     template<typename T, typename p, size_t n>
03195     struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)>> {
03196         using type = typename T::template mul_t<
03197             typename pow<T, p, n/2>::type,
03198             typename pow<T, p, n/2>::type
03199         >;
03200     };
03201
03202     template<typename T, typename p, size_t n>
03203     struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)>> {
03204         using type = typename T::template mul_t<
03205             p,
03206             typename T::template mul_t<
03207                 typename pow<T, p, n/2>::type,
03208                 typename pow<T, p, n/2>::type
03209             >
03210         >;
03211     };
03212
03213     template<typename T, typename p, size_t n>
03214     struct pow<T, p, n, std::enable_if_t<n == 0>> { using type = typename T::one; };
03215 } // namespace internal
03216
03217 template<typename T, typename p, size_t n>
03218 using pow_t = typename internal::pow<T, p, n>::type;
03219
03220 template<typename T, typename p, size_t n>
03221 static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
03222
03223 template<typename T, size_t p>
03224 static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
    internal::pow_scalar<T>::template func<p>(x); }
03225
03226 namespace internal {
03227     template<typename, template<typename, size_t> typename, class>
03228     struct make_taylor_impl;
03229
03230     template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
03231     struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...>> {

```

```

03248         using type = typename polynomial<FractionField<T>::template val<typename coeff_at<T,
Is::type...>;
03249     };
03250 }
03251
03256 template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
03257 using Taylor = typename internal::make_taylor_impl<
03258     T,
03259     coeff_at,
03260     internal::make_index_sequence_reverse<deg + 1>::type>;
03261
03262 namespace internal {
03263     template<typename T, size_t i>
03264     struct exp_coeff {
03265         using type = makefraction_t<T, typename T::one, factorial_t<T, i>;>;
03266     };
03267
03268     template<typename T, size_t i, typename E = void>
03269     struct sin_coeff_helper {};
03270
03271     template<typename T, size_t i>
03272     struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03273         using type = typename FractionField<T>::zero;
03274     };
03275
03276     template<typename T, size_t i>
03277     struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03278         using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>;>;
03279     };
03280
03281     template<typename T, size_t i>
03282     struct sh_coeff {
03283         using type = typename sin_coeff_helper<T, i>::type;
03284     };
03285
03286     template<typename T, size_t i, typename E = void>
03287     struct sh_coeff_helper {};
03288
03289     template<typename T, size_t i>
03290     struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03291         using type = typename FractionField<T>::zero;
03292     };
03293
03294     template<typename T, size_t i>
03295     struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03296         using type = makefraction_t<T, typename T::one, factorial_t<T, i>;>;
03297     };
03298
03299     template<typename T, size_t i>
03300     struct sh_coeff {
03301         using type = typename sh_coeff_helper<T, i>::type;
03302     };
03303
03304     template<typename T, size_t i, typename E = void>
03305     struct cos_coeff_helper {};
03306
03307     template<typename T, size_t i>
03308     struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03309         using type = typename FractionField<T>::zero;
03310     };
03311
03312     template<typename T, size_t i>
03313     struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03314         using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>;>;
03315     };
03316
03317     template<typename T, size_t i>
03318     struct cos_coeff {
03319         using type = typename cos_coeff_helper<T, i>::type;
03320     };
03321
03322     template<typename T, size_t i, typename E = void>
03323     struct cosh_coeff_helper {};
03324
03325     template<typename T, size_t i>
03326     struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03327         using type = typename FractionField<T>::zero;
03328     };
03329
03330     template<typename T, size_t i>
03331     struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03332         using type = makefraction_t<T, typename T::one, factorial_t<T, i>;>;
03333     };
03334
03335     template<typename T, size_t i>
03336     struct cosh_coeff {
03337         using type = typename cosh_coeff_helper<T, i>::type;

```

```

03338     };
03339
03340     template<typename T, size_t i>
03341     struct geom_coeff { using type = typename FractionField<T>::one; };
03342
03343
03344     template<typename T, size_t i, typename E = void>
03345     struct atan_coeff_helper;
03346
03347     template<typename T, size_t i>
03348     struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03349         using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;
03350     };
03351
03352     template<typename T, size_t i>
03353     struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03354         using type = typename FractionField<T>::zero;
03355     };
03356
03357     template<typename T, size_t i>
03358     struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
03359
03360     template<typename T, size_t i, typename E = void>
03361     struct asin_coeff_helper;
03362
03363     template<typename T, size_t i>
03364     struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03365         using type = makefraction_t<T,
03366             factorial_t<T, i - 1>,
03367             typename T::template mul_t<
03368                 typename T::template val<i>,
03369                 T::template mul_t<
03370                     pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
03371                     pow<T, factorial_t<T, i / 2>, 2
03372                 >,
03373                 >
03374             >;
03375     };
03376
03377     template<typename T, size_t i>
03378     struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03379         using type = typename FractionField<T>::zero;
03380     };
03381
03382     template<typename T, size_t i>
03383     struct asin_coeff {
03384         using type = typename asin_coeff_helper<T, i>::type;
03385     };
03386
03387     template<typename T, size_t i>
03388     struct lnpl_coeff {
03389         using type = makefraction_t<T,
03390             alternate_t<T, i + 1>,
03391             typename T::template val<i>;
03392     };
03393
03394     template<typename T>
03395     struct lnpl_coeff<T, 0> { using type = typename FractionField<T>::zero; };
03396
03397     template<typename T, size_t i, typename E = void>
03398     struct asinh_coeff_helper;
03399
03400     template<typename T, size_t i>
03401     struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03402         using type = makefraction_t<T,
03403             typename T::template mul_t<
03404                 alternate_t<T, i / 2>,
03405                 factorial_t<T, i - 1>
03406             >,
03407             typename T::template mul_t<
03408                 typename T::template mul_t<
03409                     typename T::template val<i>,
03410                     pow_t<T, factorial_t<T, i / 2>, 2>
03411                 >,
03412                 pow_t<T, typename T::template inject_constant_t<4>, i / 2>
03413             >
03414             >;
03415     };
03416
03417     template<typename T, size_t i>
03418     struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03419         using type = typename FractionField<T>::zero;
03420     };
03421
03422     template<typename T, size_t i>
03423     struct asinh_coeff {
03424         using type = typename asinh_coeff_helper<T, i>::type;

```

```

03425     };
03426
03427     template<typename T, size_t i, typename E = void>
03428     struct atanh_coeff_helper;
03429
03430     template<typename T, size_t i>
03431     struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
03432         // 1/i
03433         using type = typename FractionField<T>::template val<
03434             typename T::one,
03435             typename T::template inject_constant_t<i>;
03436     };
03437
03438     template<typename T, size_t i>
03439     struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
03440         using type = typename FractionField<T>::zero;
03441     };
03442
03443     template<typename T, size_t i>
03444     struct atanh_coeff {
03445         using type = typename atanh_coeff_helper<T, i>::type;
03446     };
03447
03448     template<typename T, size_t i, typename E = void>
03449     struct tan_coeff_helper;
03450
03451     template<typename T, size_t i>
03452     struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0> {
03453         using type = typename FractionField<T>::zero;
03454     };
03455
03456     template<typename T, size_t i>
03457     struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0> {
03458     private:
03459         // 4^((i+1)/2)
03460         using _4p = typename FractionField<T>::template inject_t<
03461             pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2>;
03462         // 4^((i+1)/2) - 1
03463         using _4pml = typename FractionField<T>::template
03464             sub_t<_4p, typename FractionField<T>::one>;
03465         // (-1)^((i-1)/2)
03466         using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2>;
03467         using dividend = typename FractionField<T>::template mul_t<
03468             altp,
03469             FractionField<T>::template mul_t<
03470                 _4p,
03471                 FractionField<T>::template mul_t<
03472                     _4pml,
03473                     bernoulli_t<T, (i + 1)>
03474                 >
03475             >
03476     public:
03477         using type = typename FractionField<T>::template div_t<dividend,
03478             typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03479     };
03480
03481     template<typename T, size_t i>
03482     struct tan_coeff {
03483         using type = typename tan_coeff_helper<T, i>::type;
03484     };
03485
03486     template<typename T, size_t i, typename E = void>
03487     struct tanh_coeff_helper;
03488
03489     template<typename T, size_t i>
03490     struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0> {
03491         using type = typename FractionField<T>::zero;
03492     };
03493
03494     template<typename T, size_t i>
03495     struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0> {
03496     private:
03497         using _4p = typename FractionField<T>::template inject_t<
03498             pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2>;
03499         using _4pml = typename FractionField<T>::template
03500             sub_t<_4p, typename FractionField<T>::one>;
03501         using dividend =
03502             typename FractionField<T>::template mul_t<
03503                 _4p,
03504                 typename FractionField<T>::template mul_t<
03505                     _4pml,
03506                     bernoulli_t<T, (i + 1)>>::type;
03507     public:
03508         using type = typename FractionField<T>::template div_t<dividend,
03509             FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03510     };

```

```

03510
03511     template<typename T, size_t i>
03512     struct tanh_coeff {
03513         using type = typename tanh_coeff_helper<T, i>::type;
03514     };
03515 } // namespace internal
03516
03520     template<typename Integers, size_t deg>
03521     using exp = taylor<Integers, internal::exp_coeff, deg>;
03522
03526     template<typename Integers, size_t deg>
03527     using expm1 = typename polynomial<FractionField<Integers>>::template sub_t<
03528         exp<Integers, deg>,
03529         typename polynomial<FractionField<Integers>>::one>;
03530
03534     template<typename Integers, size_t deg>
03535     using lnpl = taylor<Integers, internal::lnpl_coeff, deg>;
03536
03540     template<typename Integers, size_t deg>
03541     using atan = taylor<Integers, internal::atan_coeff, deg>;
03542
03546     template<typename Integers, size_t deg>
03547     using sin = taylor<Integers, internal::sin_coeff, deg>;
03548
03552     template<typename Integers, size_t deg>
03553     using sinh = taylor<Integers, internal::sh_coeff, deg>;
03554
03559     template<typename Integers, size_t deg>
03560     using cosh = taylor<Integers, internal::cosh_coeff, deg>;
03561
03566     template<typename Integers, size_t deg>
03567     using cos = taylor<Integers, internal::cos_coeff, deg>;
03568
03573     template<typename Integers, size_t deg>
03574     using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
03575
03580     template<typename Integers, size_t deg>
03581     using asin = taylor<Integers, internal::asin_coeff, deg>;
03582
03587     template<typename Integers, size_t deg>
03588     using asinh = taylor<Integers, internal::asinh_coeff, deg>;
03589
03594     template<typename Integers, size_t deg>
03595     using atanh = taylor<Integers, internal::atanh_coeff, deg>;
03596
03601     template<typename Integers, size_t deg>
03602     using tan = taylor<Integers, internal::tan_coeff, deg>;
03603
03608     template<typename Integers, size_t deg>
03609     using tanh = taylor<Integers, internal::tanh_coeff, deg>;
03610 } // namespace aerobus
03611
03612 // continued fractions
03613 namespace aerobus {
03614     template<int64_t... values>
03615     struct ContinuedFraction {};
03616
03621     template<int64_t a0>
03622     struct ContinuedFraction<a0> {
03623         using type = typename q64::template inject_constant_t<a0>;
03624         static constexpr double val = static_cast<double>(a0);
03625     };
03626
03632     template<int64_t a0, int64_t... rest>
03633     struct ContinuedFraction<a0, rest...> {
03634         using type = q64::template add_t<
03635             typename q64::template inject_constant_t<a0>,
03636             typename q64::template div_t<
03637                 typename q64::one,
03638                 typename ContinuedFraction<rest...>::type
03639             >;
03640
03641         static constexpr double val = type::template get<double>();
03642     };
03643
03649     using PI_fraction =
03650     ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1>;
03651     using E_fraction =
03652     ContinuedFraction<2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1>;
03653     using SQRT2_fraction =
03654     ContinuedFraction<1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2>;
03655     using SQRT3_fraction =
03656     ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2>;
03657 // NOLINT
03658 } // namespace aerobus
03659
03657
03658 // known polynomials

```

```

03659 namespace aerobus {
03660     // CChebyshev
03661     namespace internal {
03662         template<int kind, size_t deg, typename I>
03663         struct chebyshev_helper {
03664             using type = typename polynomial<I>::template sub_t<
03665                 typename polynomial<I>::template mul_t<
03666                     typename polynomial<I>::template mul_t<
03667                         typename polynomial<I>::template inject_constant_t<2>,
03668                         typename polynomial<I>::X>,
03669                         typename chebyshev_helper<kind, deg - 1, I>::type
03670                     >,
03671                     typename chebyshev_helper<kind, deg - 2, I>::type
03672                 >;
03673         };
03674
03675         template<typename I>
03676         struct chebyshev_helper<1, 0, I> {
03677             using type = typename polynomial<I>::one;
03678         };
03679
03680         template<typename I>
03681         struct chebyshev_helper<1, 1, I> {
03682             using type = typename polynomial<I>::X;
03683         };
03684
03685         template<typename I>
03686         struct chebyshev_helper<2, 0, I> {
03687             using type = typename polynomial<I>::one;
03688         };
03689
03690         template<typename I>
03691         struct chebyshev_helper<2, 1, I> {
03692             using type = typename polynomial<I>::template mul_t<
03693                 typename polynomial<I>::template inject_constant_t<2>,
03694                 typename polynomial<I>::X>;
03695         };
03696     } // namespace internal
03697
03698     // Laguerre
03699     namespace internal {
03700         template<size_t deg, typename I>
03701         struct laguerre_helper {
03702             using Q = FractionField<I>;
03703             using PQ = polynomial<Q>;
03704
03705             private:
03706                 // Lk = (1 / k) * ((2 * k - 1 - x) * lkm1 - (k - 2)lkm2)
03707                 using lnm2 = typename laguerre_helper<deg - 2, I>::type;
03708                 using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03709                 // -x + 2k-1
03710                 using p = typename PQ::template val<
03711                     typename Q::template inject_constant_t<-1>,
03712                     typename Q::template inject_constant_t<2 * deg - 1>;
03713                 // 1/n
03714                 using factor = typename PQ::template inject_ring_t<
03715                     typename Q::template val<typename I::one, typename I::template
inject_constant_t<deg>>;
03716
03717             public:
03718                 using type = typename PQ::template mul_t <
03719                     factor,
03720                     typename PQ::template sub_t<
03721                         typename PQ::template mul_t<
03722                             p,
03723                             lnm1
03724                         >,
03725                         typename PQ::template mul_t<
03726                             typename PQ::template inject_constant_t<deg-1>,
03727                             lnm2
03728                         >
03729                     >
03730                 >;
03731         };
03732
03733         template<typename I>
03734         struct laguerre_helper<0, I> {
03735             using type = typename polynomial<FractionField<I>::one>;
03736         };
03737
03738         template<typename I>
03739         struct laguerre_helper<1, I> {
03740             private:
03741                 using PQ = polynomial<FractionField<I>::one>;
03742             public:
03743                 using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
03744         };

```



```

03745     } // namespace internal
03746
03747     // Bernstein
03748     namespace internal {
03749         template<size_t i, size_t m, typename I, typename E = void>
03750         struct bernstein_helper {};
03751
03752         template<typename I>
03753         struct bernstein_helper<0, 0, I> {
03754             using type = typename polynomial<I>::one;
03755         };
03756
03757         template<size_t i, size_t m, typename I>
03758         struct bernstein_helper<i, m, I, std::enable_if_t<
03759             (m > 0) && (i == 0)>> {
03760             private:
03761                 using P = polynomial<I>;
03762             public:
03763                 using type = typename P::template mul_t<
03764                     typename P::template sub_t<typename P::one, typename P::X>,
03765                     typename bernstein_helper<i, m-1, I>::type>;
03766         };
03767
03768         template<size_t i, size_t m, typename I>
03769         struct bernstein_helper<i, m, I, std::enable_if_t<
03770             (m > 0) && (i == m)>> {
03771             private:
03772                 using P = polynomial<I>;
03773             public:
03774                 using type = typename P::template mul_t<
03775                     typename P::X,
03776                     typename bernstein_helper<i-1, m-1, I>::type>;
03777         };
03778
03779         template<size_t i, size_t m, typename I>
03780         struct bernstein_helper<i, m, I, std::enable_if_t<
03781             (m > 0) && (i > 0) && (i < m)>> {
03782             private:
03783                 using P = polynomial<I>;
03784             public:
03785                 using type = typename P::template add_t<
03786                     typename P::template mul_t<
03787                         typename P::template sub_t<typename P::one, typename P::X>,
03788                         typename bernstein_helper<i, m-1, I>::type>,
03789                     typename P::template mul_t<
03790                         typename P::X,
03791                         typename bernstein_helper<i-1, m-1, I>::type>;
03792         };
03793     } // namespace internal
03794
03795     // AllOne polynomials
03796     namespace internal {
03797         template<size_t deg, typename I>
03798         struct AllOneHelper {
03799             using type = aerobus::add_t<
03800                 typename polynomial<I>::one,
03801                 typename aerobus::mul_t<
03802                     typename polynomial<I>::X,
03803                     typename AllOneHelper<deg-1, I>::type
03804                 >>;
03805         };
03806
03807         template<typename I>
03808         struct AllOneHelper<0, I> {
03809             using type = typename polynomial<I>::one;
03810         };
03811     } // namespace internal
03812
03813     // Bessel polynomials
03814     namespace internal {
03815         template<size_t deg, typename I>
03816         struct BesselHelper {
03817             private:
03818                 using P = polynomial<I>;
03819                 using factor = typename P::template monomial_t<
03820                     typename I::template inject_constant_t<(2*deg - 1)>,
03821                     1>;
03822             public:
03823                 using type = typename P::template add_t<
03824                     typename P::template mul_t<
03825                         factor,
03826                         typename BesselHelper<deg-1, I>::type
03827                     >,
03828                     typename BesselHelper<deg-2, I>::type
03829                 >;
03830         };
03831     }

```

```

03832     template<typename I>
03833     struct BesselHelper<0, I> {
03834         using type = typename polynomial<I>::one;
03835     };
03836
03837     template<typename I>
03838     struct BesselHelper<1, I> {
03839     private:
03840         using P = polynomial<I>;
03841     public:
03842         using type = typename P::template add_t<
03843             typename P::one,
03844             typename P::X
03845         >;
03846     };
03847 } // namespace internal
03848
03849 namespace known_polynomials {
03850     enum hermite_kind {
03851         probabilist,
03852         physicist
03853     };
03854 }
03855
03856 // hermite
03857 namespace internal {
03858     template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03859     struct hermite_helper {};
03860
03861     template<size_t deg, typename I>
03862     struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
03863     private:
03864         using hnm1 = typename hermite_helper<deg - 1,
03865             known_polynomials::hermite_kind::probabilist, I>::type;
03866         using hnm2 = typename hermite_helper<deg - 2,
03867             known_polynomials::hermite_kind::probabilist, I>::type;
03868     public:
03869         using type = typename polynomial<I>::template sub_t<
03870             typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnm1>,
03871             typename polynomial<I>::template mul_t<
03872                 typename polynomial<I>::template inject_constant_t<deg - 1>,
03873                 hnm2
03874             >
03875         >;
03876     };
03877
03878     template<size_t deg, typename I>
03879     struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03880     private:
03881         using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
03882             I>::type;
03883         using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
03884             I>::type;
03885     public:
03886         using type = typename polynomial<I>::template sub_t<
03887             // 2X Hn-1
03888             typename polynomial<I>::template mul_t<
03889                 typename pi64::val<typename I::template inject_constant_t<2>,
03890                 typename I::zero>, hnm1>,
03891                 typename polynomial<I>::template mul_t<
03892                     typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03893                     hnm2
03894                 >
03895             >
03896         >;
03897     };
03898
03899     template<typename I>
03900     struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03901         using type = typename polynomial<I>::one;
03902     };
03903
03904     template<typename I>
03905     struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03906         using type = typename polynomial<I>::X;
03907     };
03908
03909     template<typename I>
03910     struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03911         using type = typename pi64::one;
03912     };
03913
03914     template<typename I>
03915     struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03916         // 2X

```

```

03918         using type = typename polynomial<I>::template val<
03919             typename I::template inject_constant_t<2>,
03920             typename I::zero>;
03921     };
03922 } // namespace internal
03923
03924 // legendre
03925 namespace internal {
03926     template<size_t n, typename I>
03927     struct legendre_helper {
03928     private:
03929         using Q = FractionField<I>;
03930         using PQ = polynomial<Q>;
03931         // 1/n constant
03932         // (2n-1)/n X
03933         using fact_left = typename PQ::template monomial_t<
03934             makefraction_t<I,
03935                 typename I::template inject_constant_t<2*n-1>,
03936                 typename I::template inject_constant_t<n>
03937             >,
03938             1>;
03939         // (n-1) / n
03940         using fact_right = typename PQ::template val<
03941             makefraction_t<I,
03942                 typename I::template inject_constant_t<n-1>,
03943                 typename I::template inject_constant_t<n>>;
03944
03945     public:
03946         using type = PQ::template sub_t<
03947             typename PQ::template mul_t<
03948                 fact_left,
03949                 typename legendre_helper<n-1, I>::type
03950             >,
03951             typename PQ::template mul_t<
03952                 fact_right,
03953                 typename legendre_helper<n-2, I>::type
03954             >
03955         >;
03956     };
03957
03958     template<typename I>
03959     struct legendre_helper<0, I> {
03960         using type = typename polynomial<FractionField<I>::one>;
03961     };
03962
03963     template<typename I>
03964     struct legendre_helper<1, I> {
03965         using type = typename polynomial<FractionField<I>::X>;
03966     };
03967 } // namespace internal
03968
03969 // bernoulli polynomials
03970 namespace internal {
03971     template<size_t n>
03972     struct bernoulli_coeff {
03973         template<typename T, size_t i>
03974         struct inner {
03975         private:
03976             using F = FractionField<T>;
03977         public:
03978             using type = typename F::template mul_t<
03979                 typename F::template inject_ring_t<combination_t<T, i, n>,
03980                 bernoulli_t<T, n-i>
03981             >;
03982         };
03983     };
03984 } // namespace internal
03985
03986 namespace internal {
03987     template<size_t n>
03988     struct touchard_coeff {
03989         template<typename T, size_t i>
03990         struct inner {
03991             using type = stirling_2_t<T, n, i>;
03992         };
03993     };
03994 } // namespace internal
03995
03996 namespace internal {
03997     template<typename I = aerobus::i64>
03998     struct AbelHelper {
03999     private:
04000         using P = aerobus::polynomial<I>;
04001
04002     public:
04003         // to keep recursion working, we need to operate on a*n and not just a
04004         template<size_t deg, I::inner_type an>

```

```

04005     struct Inner {
04006         // abel(n, a) = (x-an) * abel(n-1, a)
04007         using type = typename aerobus::mul_t<
04008             typename Inner<deg-1, an>::type,
04009             typename aerobus::sub_t<typename P::X, typename P::template inject_constant_t<an>>
04010         >;
04011     };
04012
04013     // abel(0, a) = 1
04014     template<I::inner_type an>
04015     struct Inner<0, an> {
04016         using type = P::one;
04017     };
04018
04019     // abel(1, a) = X
04020     template<I::inner_type an>
04021     struct Inner<1, an> {
04022         using type = P::X;
04023     };
04024 }
04025 } // namespace internal
04026
04027 namespace known_polynomials {
04028
04029     template<size_t n, auto a, typename I = aerobus::i64>
04030     using abel = typename internal::AbelHelper<I>::template Inner<n, a*n>::type;
04031
04032     template<size_t deg, typename I = aerobus::i64>
04033     using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
04034
04035     template<size_t deg, typename I = aerobus::i64>
04036     using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
04037
04038     template<size_t deg, typename I = aerobus::i64>
04039     using laguerre = typename internal::laguerre_helper<deg, I>::type;
04040
04041     template<size_t deg, typename I = aerobus::i64>
04042     using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
04043 I>::type;
04044
04045     template<size_t deg, typename I = aerobus::i64>
04046     using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
04047
04048     template<size_t i, size_t m, typename I = aerobus::i64>
04049     using bernstein = typename internal::bernstein_helper<i, m, I>::type;
04050
04051     template<size_t deg, typename I = aerobus::i64>
04052     using legendre = typename internal::legendre_helper<deg, I>::type;
04053
04054     template<size_t deg, typename I = aerobus::i64>
04055     using bernoulli = typename internal::bernoulli_helper<deg>::template inner, deg>;
04056
04057     template<size_t deg, typename I = aerobus::i64>
04058     using allone = typename internal::AllOneHelper<deg, I>::type;
04059
04060     template<size_t deg, typename I = aerobus::i64>
04061     using bessel = typename internal::BesselHelper<deg, I>::type;
04062
04063     template<size_t deg, typename I = aerobus::i64>
04064     using touchard = typename internal::touchard_helper<deg>::template inner, deg>;
04065 } // namespace known_polynomials
04066 } // namespace aerobus
04067
04068 #ifdef AEROBUS_CONWAY_IMPORTS
04069 // conway polynomials
04070 namespace aerobus {
04071     template<int p, int n>
04072     struct ConwayPolynomial {};
04073
04074 #ifndef DO_NOT_DOCUMENT
04075     #define ZPZV ZPZ::template val
04076     #define POLYV aerobus::polynomial<ZPZ>::template val
04077     template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<1>; }; // NOLINT
04078     template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>; }; // NOLINT
04079     template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>; }; // NOLINT
04080     template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; }; // NOLINT
04081     template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>; }; // NOLINT
04082     template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>; }; // NOLINT
04083     template<> struct ConwayPolynomial<2, 7> { using ZPZ = aerobus::zpz<2>; using type =

```

Generated by Doxygen

[illegible]

Generated by Doxygen

[illegible]

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

[illegible]

Generated by Doxygen

[illegible]

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

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

[illegible]

[illegible]

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_
00050 #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>
```

```
using F = aerobus::taylor<aerobus::i64, my_coeff, deg>;

int main() {
    constexpr double x = F<15>::eval(0.1);
    double xx = std::exp(std::exp(0.1) - 1);
    std::cout << std::setprecision(18) << x << " == " << xx << std::endl;
}
```

10.3 examples/fp16.cu

How to leverage CUDA `__half` and `__half2` 16 bits floating points number using `aerobus::i16` Warning : due to an NVIDIA bug (lack of `constexpr` operators), performance is not good

// TO compile with `nvcc -O3 -std=c++20 -arch=sm_90 fp16.cu`
 // Beforehand, you need to modify `cuda_fp16.h` by adding `__CUDA_FP16_CONSTEXPR__` to line 5039 (version 12.6)
 #include <stdio>

```
#define WITH_CUDA_FP16
#include "../src/aerobus.h"
```

```
/*
You may want to change int_type to aerobus::i32 (or i64) and float_type to float (resp. double)
*/
using int_type = aerobus::i16;
using float_type = __half2;
```

```
constexpr size_t N = 1 << 26;
```

```
template<typename T>
struct ExpmlDegree;
```

```
template<>
struct ExpmlDegree<double> {
    static constexpr size_t val = 18;
};
```

```
template<>
struct ExpmlDegree<float> {
    static constexpr size_t val = 11;
};
```

```
template<>
struct ExpmlDegree<__half2> {
    static constexpr size_t val = 6;
};
```

```
template<>
struct ExpmlDegree<__half> {
    static constexpr size_t val = 6;
};
```

```
double rand(double min, double max) {
    double range = (max - min);
    double div = RAND_MAX / range;
    return min + (rand() / div); // NOLINT
}
```

```
template<typename T>
struct GetRandT;
```

```
template<>
struct GetRandT<double> {
    static double func(double min, double max) {
        return rand(min, max);
    }
};
```

```
template<>
struct GetRandT<float> {
    static float func(double min, double max) {
        return (float) rand(min, max);
    }
};
```

```
template<>
struct GetRandT<__half2> {
    static __half2 func(double min, double max) {
        return __half2(__float2half((float)rand(min, max)), __float2half((float)rand(min, max)));
    }
};
```

```
template<>
```

```

struct GetRandT<__half>
{
    static __half func(double min, double max) {
        return __float2half((float)rand(min, max));
    }
};

using EXPM1 = aerobus::expm1<int_type, Expm1Degree<float_type>::val>;

__device__ INLINED float_type f(float_type x) {
    return EXPM1::eval(x);
}

__global__ void run(size_t N, float_type* in, float_type* out) {
    for(size_t i = threadIdx.x + blockDim.x * blockIdx.x; i < N; i += blockDim.x * gridDim.x) {
        // fp16 FMA pipeline is quite wide so we need to feed it with a LOT of computations
        out[i] = f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(in[i])))))))))))))));
    }
}

#define cudaErrorCheck(ans) { gpuAssert((ans), __FILE__, __LINE__); }
inline void gpuAssert(cudaError_t code, const char *file, int line, bool abort=true)
{
    if (code != cudaSuccess)
    {
        fprintf(stderr, "GPUassert: %s %s %d\n", cudaGetErrorString(code), file, line);
        if (abort) exit(code);
    }
}

int main() {
    // configure CUDA devices
    int deviceCount;
    int device = -1;
    int maxProcCount = 0;
    cudaErrorCheck(cudaGetDeviceCount(&deviceCount));
    for(int i = 0; i < deviceCount; ++i) {
        cudaDeviceProp prop;
        cudaErrorCheck(cudaGetDeviceProperties(&prop, i));
        int procCount = prop.multiProcessorCount;
        if(procCount > maxProcCount) {
            maxProcCount = procCount;
            device = i;
        }
    }

    if(device == -1) {
        ::printf("CANNOT FIND CUDA CAPABLE DEVICE -- aborting\n");
        ::abort();
    }

    cudaErrorCheck(cudaSetDevice(device));
    int blockSize; // The launch configurator returned block size
    int minGridSize; // The minimum grid size needed to achieve the
                    // maximum occupancy for a full device launch

    cudaErrorCheck(cudaOccupancyMaxPotentialBlockSize(&minGridSize, &blockSize, &run, 0, 0));

    ::printf("configure launch bounds to %d-%d\n", minGridSize, blockSize);

    // allocate and populate memory
    float_type *d_in, *d_out;
    cudaErrorCheck(cudaMalloc<float_type>(&d_in, N * sizeof(float_type)));
    cudaErrorCheck(cudaMalloc<float_type>(&d_out, N * sizeof(float_type)));

    float_type *in = reinterpret_cast<float_type*>(malloc(N * sizeof(float_type)));
    float_type *out = reinterpret_cast<float_type*>(malloc(N * sizeof(float_type)));

    for(size_t i = 0; i < N; ++i) {
        in[i] = GetRandT<float_type>::func(-0.01, 0.01);
    }

    cudaErrorCheck(cudaMemcpy(d_in, in, N * sizeof(float_type), cudaMemcpyHostToDevice));

    // execute kernel and get memory back from device
    run<<minGridSize, blockSize>>(N, d_in, d_out);
    cudaErrorCheck(cudaPeekAtLastError());
    cudaErrorCheck(cudaMemcpy(out, d_out, N * sizeof(float_type), cudaMemcpyDeviceToHost));

    cudaErrorCheck(cudaFree(d_in));
    cudaErrorCheck(cudaFree(d_out));
}

// Example of generated SASS :

/*
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875 ;

```

```

HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2.MMA R5, R6, R5, 0.5, 0.5 ;
HFMA2 R5, R6, R5, 1, 1 ;
HFMA2.MMA R5, R6, R5, RZ ;
HFMA2 R7, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875 ;
HFMA2.MMA R7, R5, R7, 0.008331298828125, 0.008331298828125 ;
HFMA2 R7, R5, R7, 0.041656494140625, 0.041656494140625 ;
HFMA2.MMA R7, R5, R7, 0.1666259765625, 0.1666259765625 ;
HFMA2 R7, R5, R7, 0.5, 0.5 ;
HFMA2.MMA R7, R5, R7, 1, 1 ;
HFMA2 R7, R5, R7, RZ.H0_H0 ;
HFMA2.MMA R5, R7, RZ, 0.0013885498046875, 0.0013885498046875 ;
HFMA2 R5, R7, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2.MMA R5, R7, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2 R5, R7, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2.MMA R5, R7, R5, 0.5, 0.5 ;
HFMA2 R5, R7, R5, 1, 1 ;
HFMA2.MMA R5, R7, R5, RZ ;
HFMA2 R6, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875 ;
HFMA2.MMA R6, R5, R6, 0.008331298828125, 0.008331298828125 ;
HFMA2 R6, R5, R6, 0.041656494140625, 0.041656494140625 ;
HFMA2.MMA R6, R5, R6, 0.1666259765625, 0.1666259765625 ;
HFMA2 R6, R5, R6, 0.5, 0.5 ;
HFMA2.MMA R6, R5, R6, 1, 1 ;
HFMA2 R6, R5, R6, RZ.H0_H0 ;
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875 ;
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2.MMA R5, R6, R5, 0.5, 0.5 ;
HFMA2 R5, R6, R5, 1, 1 ;
HFMA2.MMA R5, R6, R5, RZ ;
HFMA2 R6, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875 ;
HFMA2.MMA R6, R5, R6, 0.008331298828125, 0.008331298828125 ;
HFMA2 R6, R5, R6, 0.041656494140625, 0.041656494140625 ;
HFMA2.MMA R6, R5, R6, 0.1666259765625, 0.1666259765625 ;
HFMA2 R6, R5, R6, 0.5, 0.5 ;
HFMA2.MMA R6, R5, R6, 1, 1 ;
HFMA2 R6, R5, R6, RZ.H0_H0 ;
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875 ;
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2.MMA R5, R6, R5, 0.5, 0.5 ;
HFMA2 R5, R6, R5, 1, 1 ;
HFMA2.MMA R5, R6, R5, RZ ;
HFMA2 R6, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875 ;
HFMA2.MMA R6, R5, R6, 0.008331298828125, 0.008331298828125 ;
HFMA2 R6, R5, R6, 0.041656494140625, 0.041656494140625 ;
HFMA2.MMA R6, R5, R6, 0.1666259765625, 0.1666259765625 ;
HFMA2 R6, R5, R6, 0.5, 0.5 ;
HFMA2.MMA R6, R5, R6, 1, 1 ;
HFMA2 R6, R5, R6, RZ.H0_H0 ;
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875 ;
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2.MMA R5, R6, R5, 0.5, 0.5 ;
HFMA2 R5, R6, R5, 1, 1 ;
HFMA2.MMA R6, R6, R5, RZ ;
HFMA2 R5, R6, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875 ;
HFMA2.MMA R5, R6, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2 R5, R6, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2.MMA R5, R6, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2 R5, R6, R5, 0.5, 0.5 ;
HFMA2.MMA R7, R6, R5, 1, 1 ;
IADD3.X R5, R8, UR11, RZ, P0, !PT ;
IADD3 R3, P0, R2, R3, RZ ;
IADD3.X R0, RZ, R0, RZ, P0, !PT ;
ISETP.GE.U32.AND P0, PT, R3, UR8, PT ;
HFMA2 R7, R6, R7, RZ.H0_H0 ;

```



```

ISETP.GE.U32.AND.EX P0, PT, R0, UR9, PT, P0 ;
STG.E desc[UR6][R4.64], R7 ;
*/

```

10.4 examples/continued_fractions.cpp

How to use `aerobus::ContinuedFraction` to get approximations of known numbers

[illegible]

10.5 examples/modular_arithmetic.cpp

How to use `aerobus::zpz` to perform computations on rational fractions with coefficients in modular rings

```
#include <iostream>
#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
// note : it's now integrated in the main library, but still serves as an example

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 Z/2Z is equal to one
    std::cout << "expected = " << 1 << std::endl;
    std::cout << "computed = " << x << std::endl;
    return 0;
}

```

10.8 examples/compensated_horner.cpp

How to use compensated horner evaluation scheme to get better accuracy when evaluating polynomials close to its roots

See also

[publication](#)

```

// run with ./generate_comp_horner.sh in this directory
// that will compile and run this sample and plot all the generated data
#include "../src/aerobus.h"

using namespace aerobus; // NOLINT

constexpr size_t NB_POINTS = 400;

template<typename P, typename T, bool compensated>
DEVICE INLINED T eval(const T& x) {

```

```

    if constexpr (compensated) {
        return P::template compensated_eval<T>(x);
    } else {
        return P::template eval<T>(x);
    }
}

template<typename T>
DEVICE T exact_large(const T& x) {
    return pow_scalar<T, 5>(0.75 - x) * pow_scalar<T, 11>(1 - x);
}

template<typename T>
DEVICE T exact_small(const T& x) {
    return pow_scalar<T, 3>(x - 1);
}

template<typename P, typename T, bool compensated>
void run(T left, T right, const char *file_name, T (*exact)(const T&)) {
    FILE *f = ::fopen(file_name, "w+");
    T step = (right - left) / NB_POINTS;
    T x = left;
    for (size_t i = 0; i <= NB_POINTS; ++i) {
        ::fprintf(f, "%e %e %e\n", x, eval<P, T, compensated>(x), exact(x));
        x += step;
    }
    ::fclose(f);
}

int main() {
    {
        // (0.75 - x)^5 * (1 - x)^11
        using P = mul_t<
            pow_t<pq64, pq64::val<
                typename q64::template inject_constant_t<-1>,
                q64::val<i64::val<3>, i64::val<4>>, 5>,
            pow_t<pq64, pq64::val<typename q64::template inject_constant_t<-1>, typename q64::one>, 11>
            >;
        using FLOAT = double;
        run<P, FLOAT, false>(0.68, 1.15, "plots/large_sample_horner.dat", &exact_large);
        run<P, FLOAT, true>(0.68, 1.15, "plots/large_sample_comp_horner.dat", &exact_large);

        run<P, FLOAT, false>(0.74995, 0.75005, "plots/first_root_horner.dat", &exact_large);
        run<P, FLOAT, true>(0.74995, 0.75005, "plots/first_root_comp_horner.dat", &exact_large);

        run<P, FLOAT, false>(0.9935, 1.0065, "plots/second_root_horner.dat", &exact_large);
        run<P, FLOAT, true>(0.9935, 1.0065, "plots/second_root_comp_horner.dat", &exact_large);
    }
    {
        // (x - 1) ^ 3
        using P = make_int_polynomial_t<i32, 1, -3, 3, -1>;

        run<P, double, false>(1-0.00005, 1+0.00005, "plots/double.dat", &exact_small);
        run<P, float, true>(1-0.00005, 1+0.00005, "plots/float_comp.dat", &exact_small);
    }
}

```


Index

abs_t
 aerobus, 20

add_t
 aerobus, 20
 aerobus::i32, 59
 aerobus::i64, 65
 aerobus::polynomial< Ring >, 74
 aerobus::Quotient< Ring, X >, 82
 aerobus::zpz< p >, 107

addfractions_t
 aerobus, 20

aerobus, 15
 abs_t, 20
 add_t, 20
 addfractions_t, 20
 aligned_malloc, 34
 alternate_t, 20
 alternate_v, 35
 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_1_signed_t, 32
 stirling_1_unsigned_t, 32
 stirling_2_t, 32
 sub_t, 33
 tan, 33
 tanh, 33
 taylor, 33
 vadd_t, 34
 vmul_t, 34

aerobus::ContinuedFraction< a0 >, 47
 type, 47
 val, 48

aerobus::ContinuedFraction< a0, rest... >, 48
 type, 49
 val, 49

aerobus::ContinuedFraction< values >, 46

aerobus::ConwayPolynomial, 49

aerobus::Embed< i32, i64 >, 51
 type, 51

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

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

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

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

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

aerobus::Embed< zpz< x >, i32 >, 55
 type, 56

aerobus::i32, 57
 add_t, 59
 div_t, 59
 eq_t, 59
 eq_v, 62
 gcd_t, 59
 gt_t, 60

- inject_constant_t, 60
- inject_ring_t, 60
- inner_type, 60
- is_euclidean_domain, 62
- is_field, 62
- lt_t, 60
- mod_t, 61
- mul_t, 61
- one, 61
- pos_t, 61
- pos_v, 62
- sub_t, 62
- zero, 62
- aerobus::i32::val< x >, 91
 - enclosing_type, 92
 - get, 92
 - is_zero_t, 92
 - to_string, 92
 - v, 92
- aerobus::i64, 64
 - add_t, 65
 - div_t, 66
 - eq_t, 66
 - eq_v, 69
 - gcd_t, 66
 - gt_t, 66
 - gt_v, 69
 - inject_constant_t, 67
 - inject_ring_t, 67
 - inner_type, 67
 - is_euclidean_domain, 69
 - is_field, 69
 - lt_t, 67
 - lt_v, 70
 - mod_t, 68
 - mul_t, 68
 - one, 68
 - pos_t, 68
 - pos_v, 70
 - sub_t, 68
 - zero, 69
- aerobus::i64::val< x >, 93
 - enclosing_type, 94
 - get, 94
 - inner_type, 94
 - is_zero_t, 94
 - to_string, 94
 - v, 95
- aerobus::internal, 36
 - index_sequence_reverse, 40
 - is_instantiation_of_v, 40
 - make_index_sequence_reverse, 40
 - type_at_t, 40
- aerobus::is_prime< n >, 71
 - value, 72
- aerobus::IsEuclideanDomain, 43
- aerobus::IsField, 43
- aerobus::IsRing, 44
- aerobus::known_polynomials, 40
 - hermite_kind, 40
 - physicist, 41
 - probabilist, 41
- aerobus::polynomial< Ring >, 72
 - add_t, 74
 - derive_t, 74
 - div_t, 74
 - eq_t, 75
 - gcd_t, 75
 - gt_t, 75
 - inject_constant_t, 76
 - inject_ring_t, 76
 - is_euclidean_domain, 80
 - is_field, 80
 - lt_t, 76
 - mod_t, 76
 - monomial_t, 77
 - mul_t, 77
 - one, 77
 - pos_t, 77
 - pos_v, 80
 - simplify_t, 79
 - sub_t, 79
 - X, 79
 - zero, 79
- aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner< index, ghost >, 49
 - func, 50
- aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner<-1, ghost >, 50
 - func, 50
- aerobus::polynomial< Ring >::horner_reduction_t< P >, 56
- aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop >, 70
 - type, 71
- aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< stop, stop >, 71
 - type, 71
- aerobus::polynomial< Ring >::val< coeffN >, 102
 - aN, 103
 - coeff_at_t, 103
 - compensated_eval, 104
 - degree, 105
 - enclosing_type, 103
 - eval, 104
 - is_zero_t, 103
 - is_zero_v, 105
 - ring_type, 104
 - strip, 104
 - to_string, 104
 - value_at_t, 104
- aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E >, 45
- aerobus::polynomial< Ring >::val< coeffN >::coeff_at<

- index, std::enable_if_t<(index < 0 || index > 0)>, 45
- type, 45
- aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index == 0)>>, 46
- type, 46
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 95
- aN, 96
- coeff_at_t, 96
- compensated_eval, 97
- degree, 99
- enclosing_type, 96
- eval, 98
- is_zero_t, 97
- is_zero_v, 99
- ring_type, 97
- strip, 97
- to_string, 98
- value_at_t, 97
- aerobus::Quotient< Ring, X >, 81
- add_t, 82
- div_t, 83
- eq_t, 83
- eq_v, 85
- inject_constant_t, 83
- inject_ring_t, 83
- is_euclidean_domain, 85
- mod_t, 84
- mul_t, 84
- one, 84
- pos_t, 84
- pos_v, 85
- zero, 85
- aerobus::Quotient< Ring, X >::val< V >, 99
- raw_t, 100
- type, 100
- aerobus::type_list< Ts >, 87
- at, 88
- concat, 88
- insert, 88
- length, 89
- push_back, 88
- push_front, 89
- remove, 89
- aerobus::type_list< Ts >::pop_front, 80
- tail, 81
- type, 81
- aerobus::type_list< Ts >::split< index >, 86
- head, 86
- tail, 86
- aerobus::type_list<>, 90
- concat, 90
- insert, 90
- length, 91
- push_back, 90
- push_front, 90
- aerobus::zpz< p >, 105
- add_t, 107
- div_t, 107
- eq_t, 108
- eq_v, 111
- gcd_t, 108
- gt_t, 108
- gt_v, 111
- inject_constant_t, 109
- inner_type, 109
- is_euclidean_domain, 111
- is_field, 111
- lt_t, 109
- lt_v, 111
- mod_t, 109
- mul_t, 109
- one, 110
- pos_t, 110
- pos_v, 112
- sub_t, 110
- zero, 110
- aerobus::zpz< p >::val< x >, 100
- enclosing_type, 101
- get, 101
- is_zero_t, 101
- is_zero_v, 102
- to_string, 101
- v, 102
- aligned_malloc
- aerobus, 34
- alternate_t
- aerobus, 20
- alternate_v
- aerobus, 35
- aN
- aerobus::polynomial< Ring >::val< coeffN >, 103
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 96
- asin
- aerobus, 21
- asinh
- aerobus, 21
- at
- aerobus::type_list< Ts >, 88
- atan
- aerobus, 21
- atanh
- aerobus, 21
- bell_t
- aerobus, 23
- bernoulli_t
- aerobus, 23
- bernoulli_v
- aerobus, 35
- coeff_at_t
- aerobus::polynomial< Ring >::val< coeffN >, 103
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 96

- combination_t
 - aerobus, 23
- combination_v
 - aerobus, 35
- compensated_eval
 - aerobus::polynomial< Ring >::val< coeffN >, 104
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 97
- concat
 - aerobus::type_list< Ts >, 88
 - aerobus::type_list<>, 90
- cos
 - aerobus, 23
- cosh
 - aerobus, 25
- degree
 - aerobus::polynomial< Ring >::val< coeffN >, 105
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 99
- derive_t
 - aerobus::polynomial< Ring >, 74
- div_t
 - aerobus, 25
 - aerobus::i32, 59
 - aerobus::i64, 66
 - aerobus::polynomial< Ring >, 74
 - aerobus::Quotient< Ring, X >, 83
 - aerobus::zpz< p >, 107
- E_fraction
 - aerobus, 25
- embed_int_poly_in_fractions_t
 - aerobus, 25
- enclosing_type
 - aerobus::i32::val< x >, 92
 - aerobus::i64::val< x >, 94
 - aerobus::polynomial< Ring >::val< coeffN >, 103
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 96
 - aerobus::zpz< p >::val< x >, 101
- eq_t
 - aerobus::i32, 59
 - aerobus::i64, 66
 - aerobus::polynomial< Ring >, 75
 - aerobus::Quotient< Ring, X >, 83
 - aerobus::zpz< p >, 108
- eq_v
 - aerobus::i32, 62
 - aerobus::i64, 69
 - aerobus::Quotient< Ring, X >, 85
 - aerobus::zpz< p >, 111
- eval
 - aerobus::polynomial< Ring >::val< coeffN >, 104
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 98
- 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
- func
 - aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner< index, ghost >, 50
 - aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner<-1, ghost >, 50
- gcd_t
 - aerobus, 27
 - aerobus::i32, 59
 - aerobus::i64, 66
 - aerobus::polynomial< Ring >, 75
 - aerobus::zpz< p >, 108
- geometric_sum
 - aerobus, 27
- get
 - aerobus::i32::val< x >, 92
 - aerobus::i64::val< x >, 94
 - aerobus::zpz< p >::val< x >, 101
- gt_t
 - aerobus::i32, 60
 - aerobus::i64, 66
 - aerobus::polynomial< Ring >, 75
 - aerobus::zpz< p >, 108
- gt_v
 - aerobus::i64, 69
 - aerobus::zpz< p >, 111
- head
 - aerobus::type_list< Ts >::split< index >, 86
- hermite_kind
 - aerobus::known_polynomials, 40
- index_sequence_reverse
 - aerobus::internal, 40
- inject_constant_t
 - aerobus::i32, 60
 - aerobus::i64, 67
 - aerobus::polynomial< Ring >, 76
 - aerobus::Quotient< Ring, X >, 83
 - aerobus::zpz< p >, 109
- inject_ring_t
 - aerobus::i32, 60
 - aerobus::i64, 67

- aerobus::polynomial< Ring >, 76
- aerobus::Quotient< Ring, X >, 83
- inner_type
 - aerobus::i32, 60
 - aerobus::i64, 67
 - aerobus::i64::val< x >, 94
 - aerobus::zpz< p >, 109
- insert
 - aerobus::type_list< Ts >, 88
 - aerobus::type_list<>, 90
- Introduction, 1
- is_euclidean_domain
 - aerobus::i32, 62
 - aerobus::i64, 69
 - aerobus::polynomial< Ring >, 80
 - aerobus::Quotient< Ring, X >, 85
 - aerobus::zpz< p >, 111
- is_field
 - aerobus::i32, 62
 - aerobus::i64, 69
 - aerobus::polynomial< Ring >, 80
 - aerobus::zpz< p >, 111
- is_instantiation_of_v
 - aerobus::internal, 40
- is_zero_t
 - aerobus::i32::val< x >, 92
 - aerobus::i64::val< x >, 94
 - aerobus::polynomial< Ring >::val< coeffN >, 103
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 97
 - aerobus::zpz< p >::val< x >, 101
- is_zero_v
 - aerobus::polynomial< Ring >::val< coeffN >, 105
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 99
 - aerobus::zpz< p >::val< x >, 102
- length
 - aerobus::type_list< Ts >, 89
 - aerobus::type_list<>, 91
- Inp1
 - aerobus, 27
- lt_t
 - aerobus::i32, 60
 - aerobus::i64, 67
 - aerobus::polynomial< Ring >, 76
 - aerobus::zpz< p >, 109
- lt_v
 - aerobus::i64, 70
 - aerobus::zpz< p >, 111
- make_frac_polynomial_t
 - aerobus, 28
- make_index_sequence_reverse
 - aerobus::internal, 40
- 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, 68
 - aerobus::polynomial< Ring >, 76
 - aerobus::Quotient< Ring, X >, 84
 - aerobus::zpz< p >, 109
- monomial_t
 - aerobus::polynomial< Ring >, 77
- mul_t
 - aerobus, 29
 - aerobus::i32, 61
 - aerobus::i64, 68
 - aerobus::polynomial< Ring >, 77
 - aerobus::Quotient< Ring, X >, 84
 - aerobus::zpz< p >, 109
- mulfractions_t
 - aerobus, 29
- one
 - aerobus::i32, 61
 - aerobus::i64, 68
 - aerobus::polynomial< Ring >, 77
 - aerobus::Quotient< Ring, X >, 84
 - aerobus::zpz< p >, 110
- physicist
 - aerobus::known_polynomials, 41
- pi64
 - aerobus, 30
- PI_fraction
 - aerobus, 30
- pos_t
 - aerobus::i32, 61
 - aerobus::i64, 68
 - aerobus::polynomial< Ring >, 77
 - aerobus::Quotient< Ring, X >, 84
 - aerobus::zpz< p >, 110
- pos_v
 - aerobus::i32, 62
 - aerobus::i64, 70
 - aerobus::polynomial< Ring >, 80
 - aerobus::Quotient< Ring, X >, 85
 - aerobus::zpz< p >, 112
- pow_t
 - aerobus, 30
- pq64
 - aerobus, 30
- probabilist
 - aerobus::known_polynomials, 41
- push_back
 - aerobus::type_list< Ts >, 88
 - aerobus::type_list<>, 90
- push_front
 - aerobus::type_list< Ts >, 89
 - aerobus::type_list<>, 90

- q32
 - aerobus, [30](#)
- q64
 - aerobus, [31](#)
- raw_t
 - aerobus::Quotient< Ring, X >::val< V >, [100](#)
- README.md, [113](#)
- remove
 - aerobus::type_list< Ts >, [89](#)
- ring_type
 - aerobus::polynomial< Ring >::val< coeffN >, [104](#)
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, [97](#)
- simplify_t
 - aerobus::polynomial< Ring >, [79](#)
- sin
 - aerobus, [31](#)
- sinh
 - aerobus, [31](#)
- SQRT2_fraction
 - aerobus, [31](#)
- SQRT3_fraction
 - aerobus, [31](#)
- src/aerobus.h, [113](#)
- src/examples.h, [207](#)
- stirling_1_signed_t
 - aerobus, [32](#)
- stirling_1_unsigned_t
 - aerobus, [32](#)
- stirling_2_t
 - aerobus, [32](#)
- strip
 - aerobus::polynomial< Ring >::val< coeffN >, [104](#)
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, [97](#)
- sub_t
 - aerobus, [33](#)
 - aerobus::i32, [62](#)
 - aerobus::i64, [68](#)
 - aerobus::polynomial< Ring >, [79](#)
 - aerobus::zpz< p >, [110](#)
- tail
 - aerobus::type_list< Ts >::pop_front, [81](#)
 - aerobus::type_list< Ts >::split< index >, [86](#)
- tan
 - aerobus, [33](#)
- tanh
 - aerobus, [33](#)
- taylor
 - aerobus, [33](#)
- to_string
 - aerobus::i32::val< x >, [92](#)
 - aerobus::i64::val< x >, [94](#)
 - aerobus::polynomial< Ring >::val< coeffN >, [104](#)
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, [98](#)
- aerobus::zpz< p >::val< x >, [101](#)
- type
 - aerobus::ContinuedFraction< a0 >, [47](#)
 - aerobus::ContinuedFraction< a0, rest... >, [49](#)
 - aerobus::Embed< i32, i64 >, [51](#)
 - aerobus::Embed< polynomial< Small >, polynomial< Large > >, [52](#)
 - aerobus::Embed< q32, q64 >, [53](#)
 - aerobus::Embed< Quotient< Ring, X >, Ring >, [54](#)
 - aerobus::Embed< Ring, FractionField< Ring > >, [55](#)
 - aerobus::Embed< zpz< x >, i32 >, [56](#)
 - aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop >, [71](#)
 - aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< stop, stop >, [71](#)
 - aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0 || index > 0)> >, [45](#)
 - aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> >, [46](#)
 - aerobus::Quotient< Ring, X >::val< V >, [100](#)
 - aerobus::type_list< Ts >::pop_front, [81](#)
- type_at_t
 - aerobus::internal, [40](#)
- v
 - aerobus::i32::val< x >, [92](#)
 - aerobus::i64::val< x >, [95](#)
 - aerobus::zpz< p >::val< x >, [102](#)
- vadd_t
 - aerobus, [34](#)
- val
 - aerobus::ContinuedFraction< a0 >, [48](#)
 - aerobus::ContinuedFraction< a0, rest... >, [49](#)
- value
 - aerobus::is_prime< n >, [72](#)
- value_at_t
 - aerobus::polynomial< Ring >::val< coeffN >, [104](#)
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, [97](#)
- vmul_t
 - aerobus, [34](#)
- X
 - aerobus::polynomial< Ring >, [79](#)
- zero
 - aerobus::i32, [62](#)
 - aerobus::i64, [69](#)
 - aerobus::polynomial< Ring >, [79](#)
 - aerobus::Quotient< Ring, X >, [85](#)
 - aerobus::zpz< p >, [110](#)