

Aerobus

v1.2

Generated by Doxygen 1.9.8



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

6.1.2.13 E_fraction	24
6.1.2.14 exp	24
6.1.2.15 expm1	24
6.1.2.16 factorial_t	24
6.1.2.17 fpq32	25
6.1.2.18 fpq64	25
6.1.2.19 FractionField	25
6.1.2.20 gcd_t	25
6.1.2.21 geometric_sum	25
6.1.2.22 lnp1	26
6.1.2.23 make_q32_t	26
6.1.2.24 make_q64_t	26
6.1.2.25 makefraction_t	26
6.1.2.26 mulfractions_t	27
6.1.2.27 pi64	27
6.1.2.28 PI_fraction	27
6.1.2.29 pow_t	27
6.1.2.30 pq64	28
6.1.2.31 q32	28
6.1.2.32 q64	28
6.1.2.33 sin	28
6.1.2.34 sinh	28
6.1.2.35 SQRT2_fraction	28
6.1.2.36 SQRT3_fraction	29
6.1.2.37 stirling_signed_t	29
6.1.2.38 stirling_unsigned_t	29
6.1.2.39 tan	29
6.1.2.40 tanh	30
6.1.2.41 taylor	30
6.1.2.42 vadd_t	30
6.1.2.43 vmul_t	30
6.1.3 Function Documentation	31
6.1.3.1 aligned_malloc()	31
6.1.3.2 field()	31
6.1.4 Variable Documentation	31
6.1.4.1 alternate_v	31
6.1.4.2 bernoulli_v	32
6.1.4.3 combination_v	32
6.1.4.4 factorial_v	32
6.2 aerobus::internal Namespace Reference	33
6.2.1 Detailed Description	36
6.2.2 Typedef Documentation	36

6.2.2.1 <code>make_index_sequence_reverse</code>	36
6.2.2.2 <code>type_at_t</code>	36
6.2.3 Function Documentation	36
6.2.3.1 <code>index_sequence_reverse()</code>	36
6.2.4 Variable Documentation	36
6.2.4.1 <code>is_instantiation_of_v</code>	36
6.3 <code>aerobus::known_polynomials</code> Namespace Reference	36
6.3.1 Detailed Description	37
6.3.2 Typedef Documentation	37
6.3.2.1 <code>bernoulli</code>	37
6.3.2.2 <code>bernstein</code>	38
6.3.2.3 <code>chebyshev_T</code>	38
6.3.2.4 <code>chebyshev_U</code>	38
6.3.2.5 <code>hermite_phys</code>	39
6.3.2.6 <code>hermite_prob</code>	39
6.3.2.7 <code>laguerre</code>	40
6.3.2.8 <code>legendre</code>	40
6.3.3 Enumeration Type Documentation	40
6.3.3.1 <code>hermite_kind</code>	40
<b>7 Concept Documentation</b>	<b>41</b>
7.1 <code>aerobus::IsEuclideanDomain</code> Concept Reference	41
7.1.1 Concept definition	41
7.1.2 Detailed Description	41
7.2 <code>aerobus::IsField</code> Concept Reference	41
7.2.1 Concept definition	41
7.2.2 Detailed Description	42
7.3 <code>aerobus::IsRing</code> Concept Reference	42
7.3.1 Concept definition	42
7.3.2 Detailed Description	42
<b>8 Class Documentation</b>	<b>43</b>
8.1 <code>aerobus::polynomial&lt; Ring &gt;::val&lt; coeffN &gt;::coeff_at&lt; index, E &gt;</code> Struct Template Reference	43
8.2 <code>aerobus::polynomial&lt; Ring &gt;::val&lt; coeffN &gt;::coeff_at&lt; index, std::enable_if_t&lt;(index&lt; 0  index &gt; 0)&gt; &gt;</code> Struct Template Reference	43
8.2.1 Member Typedef Documentation	43
8.2.1.1 <code>type</code>	43
8.3 <code>aerobus::polynomial&lt; Ring &gt;::val&lt; coeffN &gt;::coeff_at&lt; index, std::enable_if_t&lt;(index==0)&gt; &gt;</code> Struct Template Reference	44
8.3.1 Member Typedef Documentation	44
8.3.1.1 <code>type</code>	44
8.4 <code>aerobus::ContinuedFraction&lt; values &gt;</code> Struct Template Reference	44
8.5 <code>aerobus::ContinuedFraction&lt; a0 &gt;</code> Struct Template Reference	44

8.5.1 Detailed Description	44
8.5.2 Member Typedef Documentation	45
8.5.2.1 type	45
8.5.3 Member Data Documentation	45
8.5.3.1 val	45
8.6 aerobus::ContinuedFraction< a0, rest... > Struct Template Reference	45
8.6.1 Detailed Description	45
8.6.2 Member Typedef Documentation	46
8.6.2.1 type	46
8.6.3 Member Data Documentation	46
8.6.3.1 val	46
8.7 aerobus::ConwayPolynomial Struct Reference	46
8.8 aerobus::i32 Struct Reference	46
8.8.1 Detailed Description	47
8.8.2 Member Typedef Documentation	48
8.8.2.1 add_t	48
8.8.2.2 div_t	48
8.8.2.3 eq_t	48
8.8.2.4 gcd_t	48
8.8.2.5 gt_t	48
8.8.2.6 inject_constant_t	48
8.8.2.7 inject_ring_t	48
8.8.2.8 inner_type	48
8.8.2.9 lt_t	48
8.8.2.10 mod_t	48
8.8.2.11 mul_t	49
8.8.2.12 one	49
8.8.2.13 pos_t	49
8.8.2.14 sub_t	49
8.8.2.15 zero	49
8.8.3 Member Data Documentation	49
8.8.3.1 eq_v	49
8.8.3.2 is_euclidean_domain	49
8.8.3.3 is_field	50
8.8.3.4 pos_v	50
8.9 aerobus::i64 Struct Reference	50
8.9.1 Detailed Description	51
8.9.2 Member Typedef Documentation	51
8.9.2.1 add_t	51
8.9.2.2 div_t	51
8.9.2.3 eq_t	51
8.9.2.4 gcd_t	52

8.9.2.5 <code>gt_t</code>	52
8.9.2.6 <code>inject_constant_t</code>	52
8.9.2.7 <code>inject_ring_t</code>	52
8.9.2.8 <code>inner_type</code>	52
8.9.2.9 <code>lt_t</code>	52
8.9.2.10 <code>mod_t</code>	52
8.9.2.11 <code>mul_t</code>	53
8.9.2.12 <code>one</code>	53
8.9.2.13 <code>pos_t</code>	53
8.9.2.14 <code>sub_t</code>	53
8.9.2.15 <code>zero</code>	53
8.9.3 Member Data Documentation	53
8.9.3.1 <code>eq_v</code>	53
8.9.3.2 <code>gt_v</code>	53
8.9.3.3 <code>is_euclidean_domain</code>	54
8.9.3.4 <code>is_field</code>	54
8.9.3.5 <code>lt_v</code>	54
8.9.3.6 <code>pos_v</code>	54
8.10 <code>aerobus::is_prime&lt; n &gt;</code> Struct Template Reference	54
8.10.1 Detailed Description	54
8.10.2 Member Data Documentation	55
8.10.2.1 <code>value</code>	55
8.11 <code>aerobus::polynomial&lt; Ring &gt;</code> Struct Template Reference	55
8.11.1 Detailed Description	56
8.11.2 Member Typedef Documentation	57
8.11.2.1 <code>add_t</code>	57
8.11.2.2 <code>derive_t</code>	57
8.11.2.3 <code>div_t</code>	57
8.11.2.4 <code>eq_t</code>	57
8.11.2.5 <code>gcd_t</code>	58
8.11.2.6 <code>gt_t</code>	58
8.11.2.7 <code>inject_constant_t</code>	58
8.11.2.8 <code>inject_ring_t</code>	58
8.11.2.9 <code>lt_t</code>	59
8.11.2.10 <code>mod_t</code>	59
8.11.2.11 <code>monomial_t</code>	59
8.11.2.12 <code>mul_t</code>	59
8.11.2.13 <code>one</code>	60
8.11.2.14 <code>pos_t</code>	60
8.11.2.15 <code>simplify_t</code>	60
8.11.2.16 <code>sub_t</code>	60
8.11.2.17 <code>X</code>	61

8.11.2.18 zero . . . . .	61
8.11.3 Member Data Documentation . . . . .	61
8.11.3.1 is_euclidean_domain . . . . .	61
8.11.3.2 is_field . . . . .	61
8.11.3.3 pos_v . . . . .	61
8.12 aerobus::type_list< Ts >::pop_front Struct Reference . . . . .	61
8.12.1 Detailed Description . . . . .	62
8.12.2 Member Typedef Documentation . . . . .	62
8.12.2.1 tail . . . . .	62
8.12.2.2 type . . . . .	62
8.13 aerobus::Quotient< Ring, X > Struct Template Reference . . . . .	62
8.13.1 Detailed Description . . . . .	63
8.13.2 Member Typedef Documentation . . . . .	64
8.13.2.1 add_t . . . . .	64
8.13.2.2 div_t . . . . .	64
8.13.2.3 eq_t . . . . .	64
8.13.2.4 inject_constant_t . . . . .	64
8.13.2.5 inject_ring_t . . . . .	65
8.13.2.6 mod_t . . . . .	65
8.13.2.7 mul_t . . . . .	65
8.13.2.8 one . . . . .	65
8.13.2.9 pos_t . . . . .	66
8.13.2.10 zero . . . . .	66
8.13.3 Member Data Documentation . . . . .	66
8.13.3.1 eq_v . . . . .	66
8.13.3.2 is_euclidean_domain . . . . .	66
8.13.3.3 pos_v . . . . .	66
8.14 aerobus::type_list< Ts >::split< index > Struct Template Reference . . . . .	67
8.14.1 Detailed Description . . . . .	67
8.14.2 Member Typedef Documentation . . . . .	67
8.14.2.1 head . . . . .	67
8.14.2.2 tail . . . . .	67
8.15 aerobus::type_list< Ts > Struct Template Reference . . . . .	68
8.15.1 Detailed Description . . . . .	68
8.15.2 Member Typedef Documentation . . . . .	69
8.15.2.1 at . . . . .	69
8.15.2.2 concat . . . . .	69
8.15.2.3 insert . . . . .	69
8.15.2.4 push_back . . . . .	69
8.15.2.5 push_front . . . . .	70
8.15.2.6 remove . . . . .	70
8.15.3 Member Data Documentation . . . . .	70



8.15.3.1 length	70
8.16 aerobus::type_list<> Struct Reference	70
8.16.1 Detailed Description	71
8.16.2 Member Typedef Documentation	71
8.16.2.1 concat	71
8.16.2.2 insert	71
8.16.2.3 push_back	71
8.16.2.4 push_front	71
8.16.3 Member Data Documentation	72
8.16.3.1 length	72
8.17 aerobus::i32::val< x > Struct Template Reference	72
8.17.1 Detailed Description	72
8.17.2 Member Typedef Documentation	73
8.17.2.1 enclosing_type	73
8.17.2.2 is_zero_t	73
8.17.3 Member Function Documentation	73
8.17.3.1 eval()	73
8.17.3.2 get()	73
8.17.3.3 to_string()	74
8.17.4 Member Data Documentation	74
8.17.4.1 v	74
8.18 aerobus::i64::val< x > Struct Template Reference	74
8.18.1 Detailed Description	75
8.18.2 Member Typedef Documentation	75
8.18.2.1 enclosing_type	75
8.18.2.2 inner_type	75
8.18.2.3 is_zero_t	75
8.18.3 Member Function Documentation	75
8.18.3.1 eval()	75
8.18.3.2 get()	76
8.18.3.3 to_string()	76
8.18.4 Member Data Documentation	76
8.18.4.1 v	76
8.19 aerobus::polynomial< Ring >::val< coeffN, coeffs > Struct Template Reference	76
8.19.1 Detailed Description	77
8.19.2 Member Typedef Documentation	77
8.19.2.1 aN	77
8.19.2.2 coeff_at_t	78
8.19.2.3 enclosing_type	78
8.19.2.4 is_zero_t	78
8.19.2.5 strip	78
8.19.3 Member Function Documentation	78

8.19.3.1 eval()	78
8.19.3.2 to_string()	79
8.19.4 Member Data Documentation	79
8.19.4.1 degree	79
8.19.4.2 is_zero_v	79
8.20 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference	80
8.20.1 Detailed Description	80
8.20.2 Member Typedef Documentation	80
8.20.2.1 type	80
8.21 aerobus::zpz< p >::val< x > Struct Template Reference	80
8.21.1 Member Typedef Documentation	81
8.21.1.1 enclosing_type	81
8.21.1.2 is_zero_t	81
8.21.2 Member Function Documentation	81
8.21.2.1 eval()	81
8.21.2.2 get()	81
8.21.2.3 to_string()	82
8.21.3 Member Data Documentation	82
8.21.3.1 v	82
8.22 aerobus::polynomial< Ring >::val< coeffN > Struct Template Reference	82
8.22.1 Detailed Description	83
8.22.2 Member Typedef Documentation	83
8.22.2.1 aN	83
8.22.2.2 coeff_at_t	83
8.22.2.3 enclosing_type	83
8.22.2.4 is_zero_t	83
8.22.2.5 strip	84
8.22.3 Member Function Documentation	84
8.22.3.1 eval()	84
8.22.3.2 to_string()	84
8.22.4 Member Data Documentation	84
8.22.4.1 degree	84
8.22.4.2 is_zero_v	84
8.23 aerobus::zpz< p > Struct Template Reference	84
8.23.1 Detailed Description	86
8.23.2 Member Typedef Documentation	86
8.23.2.1 add_t	86
8.23.2.2 div_t	86
8.23.2.3 eq_t	86
8.23.2.4 gcd_t	87
8.23.2.5 gt_t	87
8.23.2.6 inject_constant_t	87

8.23.2.7 inner_type . . . . .	87
8.23.2.8 lt_t . . . . .	88
8.23.2.9 mod_t . . . . .	88
8.23.2.10 mul_t . . . . .	88
8.23.2.11 one . . . . .	88
8.23.2.12 pos_t . . . . .	88
8.23.2.13 sub_t . . . . .	89
8.23.2.14 zero . . . . .	89
8.23.3 Member Data Documentation . . . . .	89
8.23.3.1 eq_v . . . . .	89
8.23.3.2 gt_v . . . . .	89
8.23.3.3 is_euclidean_domain . . . . .	90
8.23.3.4 is_field . . . . .	90
8.23.3.5 lt_v . . . . .	90
8.23.3.6 pos_v . . . . .	90
<b>9 File Documentation</b>	<b>93</b>
9.1 README.md File Reference . . . . .	93
9.2 src/aerobus.h File Reference . . . . .	93
9.3 aerobus.h . . . . .	93
<b>10 Examples</b>	<b>179</b>
10.1 QuotientRing . . . . .	179
10.2 type_list . . . . .	179
10.3 i32::template . . . . .	179
10.4 i32::add_t . . . . .	180
10.5 i32::sub_t . . . . .	180
10.6 i32::mul_t . . . . .	180
10.7 i32::div_t . . . . .	180
10.8 i32::gt_t . . . . .	181
10.9 i32::eq_t . . . . .	181
10.10 i32::eq_v . . . . .	181
10.11 i32::gcd_t . . . . .	181
10.12 i32::pos_t . . . . .	182
10.13 i32::pos_v . . . . .	182
10.14 i64::template . . . . .	182
10.15 i64::add_t . . . . .	182
10.16 i64::sub_t . . . . .	183
10.17 i64::mul_t . . . . .	183
10.18 i64::div_t . . . . .	183
10.19 i64::mod_t . . . . .	183
10.20 i64::gt_t . . . . .	184
10.21 i64::lt_t . . . . .	184

---

10.22	i64::lt_v	184
10.23	i64::eq_t	184
10.24	i64::eq_v	185
10.25	i64::gcd_t	185
10.26	i64::pos_t	185
10.27	i64::pos_v	185
10.28	polynomial	186
10.29	q32::add_t	186
10.30	FractionField	186
10.31	aerobus::ContinuedFraction	186
10.32	PI_fraction::val	187
10.33	E_fraction::val	187
<b>Index</b>		<b>189</b>

# Chapter 1

## Introduction

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

Everything in `Aerobus` is expressed as types.

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

### ***Everything is expressed as types***

The library serves two main purposes :

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

It is designed to be 'quite easily' extensible.

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

## 1.1 HOW TO

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

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

### 1.1.1 Unit Test

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

Move to the top directory then :

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

Terminal should write :

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

Alternate way :

```
make tests
```

From top directory.

### 1.1.2 Benchmarks

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

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

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

results on my laptop :

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

## 1.2 Structures

### 1.2.1 Predefined discrete euclidean domains

Aerobus predefines several simple euclidean domains, such as :

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

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

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

and the following "elements" :

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

### 1.2.2 Polynomials

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

In addition, values have an evaluation function :

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

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

### 1.2.3 Known polynomials

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

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

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

- chebyshev\_T
- chebyshev\_U
- laguerre
- hermite\_prob
- hermite\_phys
- bernstein
- legendre
- bernoulli

### 1.2.4 Conway polynomials

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

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

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

### 1.2.5 Taylor series

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

They can be used and evaluated:

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

Exposed functions are:

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



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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

## 1.3 Operations

### 1.3.1 Field of fractions

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

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

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

```

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

```

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

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

```

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

```

Which also have an evaluation function, as polynomial do.

### 1.3.2 Quotient

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

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

```

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

```

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

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

## 1.4 Misc

### 1.4.1 Continued Fractions

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

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

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

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



## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">aerobus</a>	Main namespace for all publicly exposed types or functions . . . . .	17
<a href="#">aerobus::internal</a>	Internal implementations, subject to breaking changes without notice . . . . .	33
<a href="#">aerobus::known_polynomials</a>	Families of well known polynomials such as Hermite or Bernstein . . . . .	36



## Chapter 3

# Concept Index

### 3.1 Concepts

Here is a list of all concepts with brief descriptions:

<a href="#">aerobus::IsEuclideanDomain</a>	
Concept to express R is an euclidean domain . . . . .	41
<a href="#">aerobus::IsField</a>	
Concept to express R is a field . . . . .	41
<a href="#">aerobus::IsRing</a>	
Concept to express R is a Ring . . . . .	42





## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">aerobus::polynomial&lt; Ring &gt;::val&lt; coeffN &gt;::coeff_at&lt; index, E &gt;</a>	43
<a href="#">aerobus::polynomial&lt; Ring &gt;::val&lt; coeffN &gt;::coeff_at&lt; index, std::enable_if_t&lt;(index&lt; 0  index &gt; 0)&gt; &gt;</a>	43
<a href="#">aerobus::polynomial&lt; Ring &gt;::val&lt; coeffN &gt;::coeff_at&lt; index, std::enable_if_t&lt;(index==0)&gt; &gt;</a>	44
<a href="#">aerobus::ContinuedFraction&lt; values &gt;</a>	44
<a href="#">aerobus::ContinuedFraction&lt; a0 &gt;</a>	
Specialization for only one coefficient, technically just 'a0'	44
<a href="#">aerobus::ContinuedFraction&lt; a0, rest... &gt;</a>	
Specialization for multiple coefficients (strictly more than one)	45
<a href="#">aerobus::ConwayPolynomial</a>	46
<a href="#">aerobus::i32</a>	
32 bits signed integers, seen as a algebraic ring with related operations	46
<a href="#">aerobus::i64</a>	
64 bits signed integers, seen as a algebraic ring with related operations	50
<a href="#">aerobus::is_prime&lt; n &gt;</a>	
Checks if n is prime	54
<a href="#">aerobus::polynomial&lt; Ring &gt;</a>	55
<a href="#">aerobus::type_list&lt; Ts &gt;::pop_front</a>	
Removes types from head of the list	61
<a href="#">aerobus::Quotient&lt; Ring, X &gt;</a>	
Quotient ring by the principal ideal generated by 'X' With <a href="#">i32</a> as Ring and <a href="#">i32::val&lt;2&gt;</a> as X, Quotient is $\mathbb{Z}/2\mathbb{Z}$	62
<a href="#">aerobus::type_list&lt; Ts &gt;::split&lt; index &gt;</a>	
Splits list at index	67
<a href="#">aerobus::type_list&lt; Ts &gt;</a>	
Empty pure template struct to handle type list	68
<a href="#">aerobus::type_list&lt;&gt;</a>	
Specialization for empty type list	70
<a href="#">aerobus::i32::val&lt; x &gt;</a>	
Values in <a href="#">i32</a> , again represented as types	72
<a href="#">aerobus::i64::val&lt; x &gt;</a>	
Values in <a href="#">i64</a>	74
<a href="#">aerobus::polynomial&lt; Ring &gt;::val&lt; coeffN, coeffs &gt;</a>	
Values (seen as types) in polynomial ring	76
<a href="#">aerobus::Quotient&lt; Ring, X &gt;::val&lt; V &gt;</a>	
Projection values in the quotient ring	80

<a href="#">aerobus::zpz&lt; p &gt;::val&lt; x &gt;</a> . . . . .	80
<a href="#">aerobus::polynomial&lt; Ring &gt;::val&lt; coeffN &gt;</a>	
Specialization for constants . . . . .	82
<a href="#">aerobus::zpz&lt; p &gt;</a> . . . . .	84

## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

src/ <a href="#">aerobus.h</a> . . . . .	93
--	----



## 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](#)
- 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 [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](#)

## Concepts

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

## Typedefs

- `template<typename T , typename A , typename B >`  
`using gcd\_t = typename internal::gcd< T >::template type< A, B >`  
*computes the greatest common divisor of A and B*
- `template<typename... vals>`  
`using vadd\_t = typename internal::vadd< vals... >::type`  
*adds multiple values ( $v_1 + v_2 + \dots + v_n$ ) vals must have same "enclosing\_type" and "enclosing\_type" must have an `add_t` binary operator*
- `template<typename... vals>`  
`using vmul\_t = typename internal::vmul< vals... >::type`  
*multiplies multiple values ( $v_1 + v_2 + \dots + v_n$ ) vals must have same "enclosing\_type" and "enclosing\_type" must have an `mul_t` binary operator*
- `template<typename val >`  
`using abs\_t = std::conditional_t< val::enclosing_type::template pos_v< val >, val, typename val::enclosing_type::template sub_t< typename val::enclosing_type::zero, val > >`  
*computes absolute value of 'val' val must be a 'value' in a Ring satisfying 'IsEuclideanDomain' concept*
- `template<typename Ring >`  
`using FractionField = typename internal::FractionFieldImpl< Ring >::type`
- `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  $V_1/V_2$  in the field of fractions of Ring*
- `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 T , size_t i>`  
`using factorial\_t = typename internal::factorial< T, i >::type`  
*computes factorial(i), as type*
- `template<typename T , size_t k, size_t n>`  
`using combination\_t = typename internal::combination< T, k, n >::type`  
*computes binomial coefficient (k among n) as type*
- `template<typename T , size_t n>`  
`using bernoulli\_t = typename internal::bernoulli< T, n >::type`  
*nth bernoulli number as type in T*
- `template<typename T , size_t n>`  
`using bell\_t = typename internal::bell_helper< T, n >::type`  
*Bell numbers.*
- `template<typename T , int k>`  
`using alternate\_t = typename internal::alternate< T, k >::type`  
 *$(-1)^k$  as type in T*
- `template<typename T , int n, int k>`  
`using stirling\_signed\_t = typename internal::stirling_helper< T, n, k >::type`  
*Stirling number of first kind (signed) – as types.*
- `template<typename T , int n, int k>`  
`using stirling\_unsigned\_t = abs\_t< typename internal::stirling_helper< T, n, k >::type >`  
*Stirling number of first kind (unsigned) – as types.*
- `template<typename T , typename p , size_t n>`  
`using pow\_t = typename internal::pow< T, p, n >::type`  
 *$p^n$  (as 'val' type in T)*
- `template<typename T , template< typename, size_t index > typename coeff_at, size_t deg>`  
`using taylor = typename internal::make_taylor_impl< T, coeff_at, internal::make\_index\_sequence\_reverse< deg+1 > >::type`
- `template<typename Integers , size_t deg>`  
`using exp = taylor< Integers, internal::exp_coeff, deg >`  
 $e^x$
- `template<typename Integers , size_t deg>`  
`using expm1 = typename polynomial< FractionField< Integers > >::template sub_t< exp< Integers, deg >, typename polynomial< FractionField< Integers > >::one >`  
 $e^x - 1$
- `template<typename Integers , size_t deg>`  
`using lnp1 = taylor< Integers, internal::lnp1_coeff, deg >`  
 $\ln(1 + x)$
- `template<typename Integers , size_t deg>`  
`using atan = taylor< Integers, internal::atan_coeff, deg >`  
 $\arctan(x)$
- `template<typename Integers , size_t deg>`  
`using sin = taylor< Integers, internal::sin_coeff, deg >`  
 $\sin(x)$
- `template<typename Integers , size_t deg>`  
`using sinh = taylor< Integers, internal::sh_coeff, deg >`  
 $\sinh(x)$
- `template<typename Integers , size_t deg>`  
`using cosh = taylor< Integers, internal::cosh_coeff, deg >`





### 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&lt;-2&gt;</code>
------------	--

#### 6.1.2.2 addfractions\_t

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

helper type : adds two fractions

##### Template Parameters

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

#### 6.1.2.3 alternate\_t

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

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

##### Template Parameters

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

#### 6.1.2.4 asin

```
template<typename Integers , size_t deg>
```

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

$\arcsin(x)$  arc sinus

#### Template Parameters

<i>Integers</i>	Ring type (for example <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

#### 6.1.2.5 asinh

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

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

#### Template Parameters

<i>Integers</i>	Ring type (for example <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

#### 6.1.2.6 atan

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

$\arctan(x)$

#### Template Parameters

<i>Integers</i>	Ring type (for example <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

#### 6.1.2.7 atanh

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

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

#### Template Parameters

<i>Integers</i>	Ring type (for example <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

### 6.1.2.8 bell\_t

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

Bell numbers.

#### Template Parameters

<i>T</i>	ring type, such as <a href="#">aerobus::i64</a>
<i>n</i>	index

### 6.1.2.9 bernoulli\_t

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

*n*th bernoulli number as type in *T*

#### Template Parameters

<i>T</i>	Ring type ( <a href="#">i64</a> )
<i>n</i>	

### 6.1.2.10 combination\_t

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

computes binomial coefficient (k among n) as type

#### Template Parameters

<i>T</i>	Ring type ( <a href="#">i32</a> for example)
----------	--

### 6.1.2.11 cos

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

$\cos(x)$  cosinus

#### Template Parameters

<i>Integers</i>	Ring type (for example <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

### 6.1.2.12 cosh

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

$\cosh(x)$  hyperbolic cosine

#### Template Parameters

<i>Integers</i>	Ring type (for example <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

### 6.1.2.13 E\_fraction

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

### 6.1.2.14 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 <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

### 6.1.2.15 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 <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

### 6.1.2.16 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. <a href="#">i32</a> )
<i>i</i>	

#### 6.1.2.17 fpq32

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

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

#### 6.1.2.18 fpq64

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

polynomial with 64 bits rational coefficients

#### 6.1.2.19 FractionField

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

#### 6.1.2.20 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)
----------	--------------------------------------

#### 6.1.2.21 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 <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

### 6.1.2.22 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 <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

### 6.1.2.23 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

### 6.1.2.24 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

### 6.1.2.25 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

## 6.1.2.26 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 FractionField<Ring>

## 6.1.2.27 pi64

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

polynomial with 64 bits integers coefficients

## 6.1.2.28 PI\_fraction

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

## 6.1.2.29 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

**6.1.2.30 pq64**

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

polynomial with 64 bits rationals coefficients

**6.1.2.31 q32**

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

32 bits rationals rationals with 32 bits numerator and denominator

**6.1.2.32 q64**

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

64 bits rationals rationals with 64 bits numerator and denominator

**6.1.2.33 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 <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

**6.1.2.34 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 <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

**6.1.2.35 SQRT2\_fraction**

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





## Template Parameters

<i>Integers</i>	Ring type (for example <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

## 6.1.2.40 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 <a href="#">i64</a> )
<i>deg</i>	taylor approximation degree

## 6.1.2.41 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 ( <a href="#">aerobus::i64</a> for example)
<i>coeff_↔ _at</i>	- implementation giving the 'value' (seen as type in FractionField<T>)
<i>deg</i>	

## 6.1.2.42 vadd\_t

```
template<typename... vals>
using aerobus::vadd\_t = typedef 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 Parameters

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

## 6.1.2.43 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, <a href="#">aerobus::i64</a> 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>	( <a href="#">aerobus::i64</a> 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>	( <a href="#">aerobus::i32</a> 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>	( <a href="#">aerobus::i64</a> for example)
<i>i</i>	

## 6.2 aerobus::internal Namespace Reference

internal implementations, subject to breaking changes without notice

### Classes

- struct **\_FractionField**
- struct **\_FractionField**< Ring, std::enable\_if\_t< Ring::is\_euclidean\_domain > >
- struct **\_is\_prime**
- struct **\_is\_prime**< 0, i >
- struct **\_is\_prime**< 1, i >
- struct **\_is\_prime**< 2, i >
- struct **\_is\_prime**< 3, i >
- struct **\_is\_prime**< 5, i >
- struct **\_is\_prime**< 7, i >
- struct **\_is\_prime**< n, i, std::enable\_if\_t<(n !=2 &&n !=3 &&n % 2 !=0 &&n % 3==0)> >
- struct **\_is\_prime**< n, i, std::enable\_if\_t<(n !=2 &&n % 2==0)> >
- struct **\_is\_prime**< n, i, std::enable\_if\_t<(n % i==0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&i \*i > n)> >
- struct **\_is\_prime**< n, i, std::enable\_if\_t<(n %(i+2) !=0 &&n % i !=0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&(i \*i<=n))> >
- struct **\_is\_prime**< n, i, std::enable\_if\_t<(n %(i+2)==0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&i \*i<=n)> >
- struct **\_is\_prime**< n, i, std::enable\_if\_t<(n >=9 &&i \*i > n)> >
- struct **alternate**
- struct **alternate**< T, k, std::enable\_if\_t< k % 2 !=0 > >
- struct **alternate**< T, k, std::enable\_if\_t< k % 2==0 > >
- struct **asin\_coeff**
- struct **asin\_coeff\_helper**
- struct **asin\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==0 > >
- struct **asin\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==1 > >
- struct **asinh\_coeff**
- struct **asinh\_coeff\_helper**
- struct **asinh\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==0 > >
- struct **asinh\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==1 > >
- struct **atan\_coeff**
- struct **atan\_coeff\_helper**
- struct **atan\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==0 > >
- struct **atan\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==1 > >
- struct **atanh\_coeff**
- struct **atanh\_coeff\_helper**
- struct **atanh\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==0 > >
- struct **atanh\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==1 > >
- struct **bell\_helper**
- struct **bell\_helper**< T, 0 >
- struct **bell\_helper**< T, 1 >
- struct **bell\_helper**< T, n, std::enable\_if\_t<(n > 1)> >
- struct **bernoulli**
- struct **bernoulli**< T, 0 >
- struct **bernoulli\_coeff**
- struct **bernoulli\_helper**
- struct **bernoulli\_helper**< T, accum, m, m >
- struct **bernstein\_helper**
- struct **bernstein\_helper**< 0, 0, l >

- struct **bernstein\_helper**< i, m, I, std::enable\_if\_t<(m > 0) &&(i > 0) &&(i < m)> >
- struct **bernstein\_helper**< i, m, I, std::enable\_if\_t<(m > 0) &&(i==0)> >
- struct **bernstein\_helper**< i, m, I, std::enable\_if\_t<(m > 0) &&(i==m)> >
- struct **chebyshev\_helper**
- struct **chebyshev\_helper**< 1, 0, I >
- struct **chebyshev\_helper**< 1, 1, I >
- struct **chebyshev\_helper**< 2, 0, I >
- struct **chebyshev\_helper**< 2, 1, I >
- struct **combination**
- struct **combination\_helper**
- struct **combination\_helper**< T, 0, n >
- struct **combination\_helper**< T, k, n, std::enable\_if\_t<(n >=0 &&k >(n/2) &&k > 0)> >
- struct **combination\_helper**< T, k, n, std::enable\_if\_t<(n >=0 &&k <=(n/2) &&k > 0)> >
- struct **cos\_coeff**
- struct **cos\_coeff\_helper**
- struct **cos\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==0> >
- struct **cos\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==1> >
- struct **cosh\_coeff**
- struct **cosh\_coeff\_helper**
- struct **cosh\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==0> >
- struct **cosh\_coeff\_helper**< T, i, std::enable\_if\_t<(i &1)==1> >
- struct **exp\_coeff**
- struct **factorial**
- struct **factorial**< T, 0 >
- struct **factorial**< T, x, std::enable\_if\_t<(x > 0)> >
- struct **FractionFieldImpl**
- struct **FractionFieldImpl**< Field, std::enable\_if\_t< Field::is\_field > >
- struct **FractionFieldImpl**< Ring, std::enable\_if\_t<!Ring::is\_field > >
- struct **gcd**

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

- struct **gcd**< Ring, std::enable\_if\_t< Ring::is\_euclidean\_domain > >
- struct **geom\_coeff**
- struct **hermite\_helper**
- struct **hermite\_helper**< 0, known\_polynomials::hermite\_kind::physicist, 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\_h**
- struct **split\_h**< 0, L1, L2 >
- struct **stirling\_helper**
- struct **stirling\_helper**< T, 0, 0 >
- struct **stirling\_helper**< T, 0, n, std::enable\_if\_t<(n > 0)> >
- struct **stirling\_helper**< T, n, 0, std::enable\_if\_t<(n > 0)> >
- struct **stirling\_helper**< T, n, k, std::enable\_if\_t<(k > 0) &&(n > 0)> >
- struct **tan\_coeff**
- struct **tan\_coeff\_helper**
- struct **tan\_coeff\_helper**< T, i, std::enable\_if\_t<(i % 2) !=0 > >
- struct **tan\_coeff\_helper**< T, i, std::enable\_if\_t<(i % 2)==0 > >
- struct **tanh\_coeff**
- struct **tanh\_coeff\_helper**
- struct **tanh\_coeff\_helper**< T, i, std::enable\_if\_t<(i % 2) !=0 > >
- struct **tanh\_coeff\_helper**< T, i, std::enable\_if\_t<(i % 2)==0 > >
- struct **type\_at**
- struct **type\_at**< 0, T, Ts... >
- struct **vadd**
- struct **vadd**< v1 >
- struct **vadd**< v1, vals... >
- struct **vmul**
- struct **vmul**< v1 >
- struct **vmul**< v1, vals... >

## Typedefs

- template<size\_t i, typename... Ts>  
using **type\_at\_t** = typename type\_at< i, Ts... >::type
- template<std::size\_t N>  
using **make\_index\_sequence\_reverse** = decltype(index\_sequence\_reverse(std::make\_index\_sequence< N >{}))

## Functions

- template<std::size\_t... Is>  
constexpr auto **index\_sequence\_reverse** (std::index\_sequence< Is... > const &) -> decltype(std::index\_↵  
sequence< sizeof...(Is) - 1U - Is... >{})

## Variables

- `template<template< typename... > typename TT, typename T >`  
`constexpr bool is\_instantiation\_of\_v = is_instantiation_of<TT, T>::value`

### 6.2.1 Detailed Description

internal implementations, subject to breaking changes without notice

### 6.2.2 Typedef Documentation

#### 6.2.2.1 `make_index_sequence_reverse`

```
template<std::size_t N>
using aerobus::internal::make\_index\_sequence\_reverse = typedef decltype(index\_sequence\_reverse(std::make_index_sequence<N>{}))
```

#### 6.2.2.2 `type_at_t`

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

### 6.2.3 Function Documentation

#### 6.2.3.1 `index_sequence_reverse()`

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

### 6.2.4 Variable Documentation

#### 6.2.4.1 `is_instantiation_of_v`

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

## 6.3 `aerobus::known_polynomials` Namespace Reference

families of well known polynomials such as Hermite or Bernstein



## Typedefs

- `template<size_t deg, typename I = aerobus::i64>`  
`using chebyshev\_T = typename internal::chebyshev_helper< 1, deg, I >::type`  
*Chebyshev polynomials of first kind.*
- `template<size_t deg, typename I = aerobus::i64>`  
`using chebyshev\_U = typename internal::chebyshev_helper< 2, deg, I >::type`  
*Chebyshev polynomials of second kind.*
- `template<size_t deg, typename I = aerobus::i64>`  
`using laguerre = typename internal::laguerre_helper< deg, I >::type`  
*Laguerre polynomials.*
- `template<size_t deg, typename I = aerobus::i64>`  
`using hermite\_prob = typename internal::hermite_helper< deg, hermite\_kind::probabilist, I >::type`  
*Hermite polynomials - probabilist form.*
- `template<size_t deg, typename I = aerobus::i64>`  
`using hermite\_phys = typename internal::hermite_helper< deg, hermite\_kind::physicist, I >::type`  
*Hermite polynomials - physicist form.*
- `template<size_t i, size_t m, typename I = aerobus::i64>`  
`using bernstein = typename internal::bernstein_helper< i, m, I >::type`  
*Bernstein polynomials.*
- `template<size_t deg, typename I = aerobus::i64>`  
`using legendre = typename internal::legendre_helper< deg, I >::type`  
*Legendre polynomials.*
- `template<size_t deg, typename I = aerobus::i64>`  
`using bernoulli = taylor< I, internal::bernoulli_coeff< deg >::template inner, deg >`  
*Bernoulli polynomials.*

## Enumerations

- `enum hermite\_kind { probabilist , physicist }`

### 6.3.1 Detailed Description

families of well known polynomials such as Hermite or Bernstein

### 6.3.2 Typedef Documentation

#### 6.3.2.1 bernoulli

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

Bernoulli polynomials.

Lives in `polynomial<FractionField<I>>`

See also

[See in Wikipedia](#)

## Template Parameters

<i>deg</i>	degree of polynomial
<i>I</i>	Integers ring (defaults to <a href="#">aerobus::i64</a> )

## 6.3.2.2 bernstein

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

Bernstein polynomials.

Lives in polynomial

## See also

[See in Wikipedia](#)

## Template Parameters

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

## 6.3.2.3 chebyshev\_T

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

Chebyshev polynomials of first kind.

## See also

[See in Wikipedia](#)

## Template Parameters

<i>deg</i>	degree of polynomial
<i>integer</i>	rings (defaults to <a href="#">aerobus::i64</a> )

## 6.3.2.4 chebyshev\_U

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

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

Chebyshev polynomials of second kind.

Lives in polynomial

*See also*

[See in Wikipedia](#)

#### Template Parameters

deg	<i>degree of polynomial</i>
integer	<i>rings (defaults to <a href="#">aerobus::i64</a>)</i>

### 6.3.2.5 hermite\_phys

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

Hermite polynomials - physicist form.

*See also*

[See in Wikipedia](#)

#### Template Parameters

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

### 6.3.2.6 hermite\_prob

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

Hermite polynomials - probabilist form.

*See also*

[See in Wikipedia](#)

#### Template Parameters

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

### 6.3.2.7 laguerre

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

Laguerre polynomials.

Lives in polynomial<FractionField<I>>

See also

[See in Wikipedia](#)

Template Parameters

<i>deg</i>	degree of polynomial
<i>I</i>	Integers ring (defaults to <a href="#">aerobus::i64</a> )

### 6.3.2.8 legendre

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

Legendre polynomials.

Lives in polynomial<FractionField<I>>

See also

[See in Wikipedia](#)

Template Parameters

<i>deg</i>	degree of polynomial
<i>I</i>	Integers Ring (defaults to <a href="#">aerobus::i64</a> )

## 6.3.3 Enumeration Type Documentation

### 6.3.3.1 hermite\_kind

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

Enumerator

probabilist	
physicist	

# Chapter 7

## Concept Documentation

### 7.1 aerobus::IsEuclideanDomain Concept Reference

Concept to express R is an euclidean domain.

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

#### 7.1.1 Concept definition

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

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

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

#### 7.1.2 Detailed Description

Concept to express R is an euclidean domain.

### 7.2 aerobus::IsField Concept Reference

Concept to express R is a field.

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

#### 7.2.1 Concept definition

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

### 7.2.2 Detailed Description

Concept to express R is a field.

## 7.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring.

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

### 7.3.1 Concept definition

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

### 7.3.2 Detailed Description

Concept to express R is a Ring.

## Chapter 8

# Class Documentation

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

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

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

- [src/aerobus.h](#)

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

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

#### Public Types

- `using type = typename Ring::zero`

#### 8.2.1 Member Typedef Documentation

##### 8.2.1.1 `type`

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

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

- [src/aerobus.h](#)

### 8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff\_at< index, std::enable\_if\_t<(index==0)> > Struct Template Reference

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

#### Public Types

- [using type = aN](#)

#### 8.3.1 Member Typedef Documentation

##### 8.3.1.1 type

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

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

- [src/aerobus.h](#)

### 8.4 aerobus::ContinuedFraction< values > Struct Template Reference

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

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::i32 Struct Reference

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

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

## Classes

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

## Public Types

- `using inner_type = int32_t`
- `using zero = val< 0 >`  
*constant zero*
- `using one = val< 1 >`  
*constant one*
- `template<auto x>`  
`using inject_constant_t = val< static_cast< int32_t >(x)>`
- `template<typename v >`  
`using inject_ring_t = v`
- `template<typename v1 , typename v2 >`  
`using add_t = typename add< v1, v2 >::type`
- `template<typename v1 , typename v2 >`  
`using sub_t = typename sub< v1, v2 >::type`
- `template<typename v1 , typename v2 >`  
`using mul_t = typename mul< v1, v2 >::type`
- `template<typename v1 , typename v2 >`  
`using div_t = typename div< v1, v2 >::type`
- `template<typename v1 , typename v2 >`  
`using mod_t = typename remainder< v1, v2 >::type`  
*modulus operator yields  $v1 \% v2$  for example : `i32::mod_t<i32::val<7>, i32::val<2>>`*
- `template<typename v1 , typename v2 >`  
`using gt_t = typename gt< v1, v2 >::type`
- `template<typename v1 , typename v2 >`  
`using lt_t = typename lt< v1, v2 >::type`
- `template<typename v1 , typename v2 >`  
`using eq_t = typename eq< v1, v2 >::type`
- `template<typename v1 , typename v2 >`  
`using gcd_t = gcd_t< i32, v1, v2 >`
- `template<typename v >`  
`using pos_t = typename pos< v >::type`

## Static Public Attributes

- `static constexpr bool is_field = false`  
*integers are not a field*
- `static constexpr bool is_euclidean_domain = true`  
*integers are an euclidean domain*
- `template<typename v1 , typename v2 >`  
`static constexpr bool eq_v = eq_t<v1, v2>::value`
- `template<typename v >`  
`static constexpr bool pos_v = pos_t<v>::value`

### 8.8.1 Detailed Description

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

## 8.8.2 Member Typedef Documentation

### 8.8.2.1 add\_t

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

### 8.8.2.2 div\_t

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

### 8.8.2.3 eq\_t

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

### 8.8.2.4 gcd\_t

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

### 8.8.2.5 gt\_t

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

### 8.8.2.6 inject\_constant\_t

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

### 8.8.2.7 inject\_ring\_t

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

### 8.8.2.8 inner\_type

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

### 8.8.2.9 lt\_t

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

### 8.8.2.10 mod\_t

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

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

## Template Parameters

<code>v1</code>	a value in <a href="#">i32</a>
<code>v2</code>	a value in <a href="#">i32</a>

**8.8.2.11 mul\_t**

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

**8.8.2.12 one**

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

constant one

**8.8.2.13 pos\_t**

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

**8.8.2.14 sub\_t**

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

**8.8.2.15 zero**

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

constant zero

**8.8.3 Member Data Documentation****8.8.3.1 eq\_v**

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

**8.8.3.2 is\_euclidean\_domain**

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

integers are an euclidean domain

### 8.8.3.3 is\_field

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

integers are not a field

### 8.8.3.4 pos\_v

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

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

- [src/aerobus.h](#)

## 8.9 aerobus::i64 Struct Reference

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

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

### Classes

- struct [val](#)  
*values in [i64](#)*

### Public Types

- [using inner\\_type = int64\\_t](#)  
*type of represented values*
- [template<auto x>](#)  
[using inject\\_constant\\_t = val< static\\_cast< int64\\_t >\(x\)>](#)
- [template<typename v >](#)  
[using inject\\_ring\\_t = v](#)  
*injects a value used for internal consistency and quotient rings implementations for example [i64::inject\\_ring\\_t<i64::val<1>>](#)  
-> [i64::val<1>](#)*
- [using zero = val< 0 >](#)  
*constant zero*
- [using one = val< 1 >](#)  
*constant one*
- [template<typename v1 , typename v2 >](#)  
[using add\\_t = typename add< v1, v2 >::type](#)
- [template<typename v1 , typename v2 >](#)  
[using sub\\_t = typename sub< v1, v2 >::type](#)
- [template<typename v1 , typename v2 >](#)  
[using mul\\_t = typename mul< v1, v2 >::type](#)
- [template<typename v1 , typename v2 >](#)  
[using div\\_t = typename div< v1, v2 >::type](#)

- `template<typename v1 , typename v2 >`  
`using mod_t = typename remainder< v1, v2 >::type`
- `template<typename v1 , typename v2 >`  
`using gt_t = typename gt< v1, v2 >::type`
- `template<typename v1 , typename v2 >`  
`using lt_t = typename lt< v1, v2 >::type`
- `template<typename v1 , typename v2 >`  
`using eq_t = typename eq< v1, v2 >::type`
- `template<typename v1 , typename v2 >`  
`using gcd_t = gcd_t< i64, v1, v2 >`
- `template<typename v >`  
`using pos_t = typename pos< v >::type`

### Static Public Attributes

- `static constexpr bool is_field = false`  
*integers are not a field*
- `static constexpr bool is_euclidean_domain = true`  
*integers are an euclidean domain*
- `template<typename v1 , typename v2 >`  
`static constexpr bool gt_v = gt_t<v1, v2>::value`  
*strictly greater operator yields v1 > v2 as boolean value*
- `template<typename v1 , typename v2 >`  
`static constexpr bool lt_v = lt_t<v1, v2>::value`
- `template<typename v1 , typename v2 >`  
`static constexpr bool eq_v = eq_t<v1, v2>::value`
- `template<typename v >`  
`static constexpr bool pos_v = pos_t<v>::value`

## 8.9.1 Detailed Description

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

## 8.9.2 Member Typedef Documentation

### 8.9.2.1 add\_t

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

### 8.9.2.2 div\_t

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

### 8.9.2.3 eq\_t

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

#### 8.9.2.4 gcd\_t

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

#### 8.9.2.5 gt\_t

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

#### 8.9.2.6 inject\_constant\_t

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

#### 8.9.2.7 inject\_ring\_t

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

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

##### Template Parameters

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

#### 8.9.2.8 inner\_type

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

type of represented values

#### 8.9.2.9 lt\_t

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

#### 8.9.2.10 mod\_t

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



### 8.9.2.11 mul\_t

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

### 8.9.2.12 one

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

constant one

### 8.9.2.13 pos\_t

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

### 8.9.2.14 sub\_t

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

### 8.9.2.15 zero

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

constant zero

## 8.9.3 Member Data Documentation

### 8.9.3.1 eq\_v

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

### 8.9.3.2 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

$v1$	: an element of <a href="#">aerobus::i64::val</a>
$v2$	: an element of <a href="#">aerobus::i64::val</a>

### 8.9.3.3 is\_euclidean\_domain

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

integers are an euclidean domain

### 8.9.3.4 is\_field

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

integers are not a field

### 8.9.3.5 lt\_v

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

### 8.9.3.6 pos\_v

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

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

- [src/aerobus.h](#)

## 8.10 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*

### 8.10.1 Detailed Description

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

checks if n is prime

## Template Parameters

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

## 8.10.2 Member Data Documentation

### 8.10.2.1 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](#)

## 8.11 aerobus::polynomial< Ring > Struct Template Reference

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

### Classes

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

### Public Types

- [using zero = val< typename Ring::zero >](#)  
*constant zero*
- [using one = val< typename Ring::one >](#)  
*constant one*
- [using X = val< typename Ring::one, typename Ring::zero >](#)  
*generator*
- [template<typename P >](#)  
[using simplify\\_t = typename simplify< P >::type](#)  
*simplifies a polynomial (recursively deletes highest degree if zero, do nothing otherwise)*
- [template<typename v1 , typename v2 >](#)  
[using add\\_t = typename add< v1, v2 >::type](#)  
*adds two polynomials*
- [template<typename v1 , typename v2 >](#)  
[using sub\\_t = typename sub< v1, v2 >::type](#)  
*subtraction of two polynomials*

- `template<typename v1 , typename v2 >`  
`using mul_t = typename mul< v1, v2 >::type`  
*multiplication of two polynomials*
- `template<typename v1 , typename v2 >`  
`using eq_t = typename eq_helper< v1, v2 >::type`  
*equality operator*
- `template<typename v1 , typename v2 >`  
`using lt_t = typename lt_helper< v1, v2 >::type`  
*strict less operator*
- `template<typename v1 , typename v2 >`  
`using gt_t = typename gt_helper< v1, v2 >::type`  
*strict greater operator*
- `template<typename v1 , typename v2 >`  
`using div_t = typename div< v1, v2 >::q_type`  
*division operator*
- `template<typename v1 , typename v2 >`  
`using mod_t = typename div_helper< v1, v2, zero, v1 >::mod_type`  
*modulo operator*
- `template<typename coeff , size_t deg>`  
`using monomial_t = typename monomial< coeff, deg >::type`  
*monomial :  $\text{coeff } X^{\text{deg}}$*
- `template<typename v >`  
`using derive_t = typename derive_helper< v >::type`  
*derivation operator*
- `template<typename v >`  
`using pos_t = typename Ring::template pos_t< typename v::aN >`  
*checks for positivity ( $an > 0$ )*
- `template<typename v1 , typename v2 >`  
`using gcd_t = std::conditional_t< Ring::is_euclidean_domain, typename make_unit< gcd_t< polynomial< Ring >, v1, v2 > >::type, void >`  
*greatest common divisor of two polynomials*
- `template<auto x>`  
`using inject_constant_t = val< typename Ring::template inject_constant_t< x > >`
- `template<typename v >`  
`using inject_ring_t = val< v >`

## Static Public Attributes

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

### 8.11.1 Detailed Description

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

polynomial with coefficients in Ring Ring must be an integral domain

## 8.11.2 Member Typedef Documentation

### 8.11.2.1 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>	

### 8.11.2.2 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>	
----------	--

### 8.11.2.3 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>	

### 8.11.2.4 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>	

**8.11.2.5 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>	

**8.11.2.6 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>	

**8.11.2.7 inject\_constant\_t**

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

**8.11.2.8 inject\_ring\_t**

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

### 8.11.2.9 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>	

### 8.11.2.10 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>	

### 8.11.2.11 monomial\_t

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

monomial :  $\text{coeff } X^{\text{deg}}$

#### Template Parameters

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

### 8.11.2.12 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>	

**8.11.2.13 one**

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

constant one

**8.11.2.14 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>	
----------	--

**8.11.2.15 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>	
----------	--

**8.11.2.16 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>	



### 8.11.2.17 X

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

generator

### 8.11.2.18 zero

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

constant zero

## 8.11.3 Member Data Documentation

### 8.11.3.1 is\_euclidean\_domain

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

### 8.11.3.2 is\_field

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

### 8.11.3.3 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

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

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

- `src/aerobus.h`

## 8.12 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*

### 8.12.1 Detailed Description

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

removes types from head of the list

### 8.12.2 Member Typedef Documentation

#### 8.12.2.1 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

#### 8.12.2.2 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](#)

## 8.13 aerobus::Quotient< Ring, X > Struct Template Reference

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

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

## Classes

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

## Public Types

- `using zero = val< typename Ring::zero >`  
*zero value*
- `using one = val< typename Ring::one >`  
*one*
- `template<typename v1 , typename v2 >`  
`using add_t = val< typename Ring::template add_t< typename v1::type, typename v2::type > >`  
*addition operator*
- `template<typename v1 , typename v2 >`  
`using mul_t = val< typename Ring::template mul_t< typename v1::type, typename v2::type > >`  
*subtraction operator*
- `template<typename v1 , typename v2 >`  
`using div_t = val< typename Ring::template div_t< typename v1::type, typename v2::type > >`  
*division operator*
- `template<typename v1 , typename v2 >`  
`using mod_t = val< typename Ring::template mod_t< typename v1::type, typename v2::type > >`  
*modulus operator*
- `template<typename v1 , typename v2 >`  
`using eq_t = typename Ring::template eq_t< typename v1::type, typename v2::type >`  
*equality operator (as type)*
- `template<typename v1 >`  
`using pos_t = std::true_type`  
*positivity operator always true*
- `template<auto x>`  
`using inject_constant_t = val< typename Ring::template inject_constant_t< x > >`
- `template<typename v >`  
`using inject_ring_t = val< v >`

## Static Public Attributes

- `template<typename v1 , typename v2 >`  
`static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value`  
*addition operator (as boolean value)*
- `template<typename v >`  
`static constexpr bool pos_v = pos_t<v>::value`  
*positivity operator always true*
- `static constexpr bool is_euclidean_domain = true`  
*quotient rings are euclidean domain*

### 8.13.1 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&lt;2&gt;</code>

## 8.13.2 Member Typedef Documentation

### 8.13.2.1 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

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

### 8.13.2.2 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

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

### 8.13.2.3 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

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

### 8.13.2.4 inject\_constant\_t

```
template<typename Ring , typename X >
```

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

### 8.13.2.5 inject\_ring\_t

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

### 8.13.2.6 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

### 8.13.2.7 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

### 8.13.2.8 one

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

one

### 8.13.2.9 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
-----------	--------------------------

### 8.13.2.10 zero

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

zero value

## 8.13.3 Member Data Documentation

### 8.13.3.1 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

### 8.13.3.2 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

### 8.13.3.3 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](#)

## 8.14 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

### 8.14.1 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>	
--------------	--

### 8.14.2 Member Typedef Documentation

#### 8.14.2.1 head

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

#### 8.14.2.2 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](#)

## 8.15 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*

#### 8.15.1 Detailed Description

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

Empty pure template struct to handle type list.



## 8.15.2 Member Typedef Documentation

### 8.15.2.1 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>	
--------------	--

### 8.15.2.2 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>	
----------	--

### 8.15.2.3 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>	

### 8.15.2.4 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>	
----------	--

**8.15.2.5 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>	
----------	--

**8.15.2.6 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>	
--------------	--

**8.15.3 Member Data Documentation****8.15.3.1 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](#)

**8.16 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

### 8.16.1 Detailed Description

specialization for empty type list

### 8.16.2 Member Typedef Documentation

#### 8.16.2.1 concat

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

#### 8.16.2.2 insert

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

#### 8.16.2.3 push\_back

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

#### 8.16.2.4 push\_front

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

### 8.16.3 Member Data Documentation

#### 8.16.3.1 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

## 8.17 aerobus::i32::val< x > Struct Template Reference

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

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

### Public Types

- [using enclosing\\_type = i32](#)  
*Enclosing ring type.*
- [using is\\_zero\\_t = std::bool\\_constant< x==0 >](#)  
*is value zero*

### Static Public Member Functions

- [template<typename valueType >](#)  
[static constexpr INLINED DEVICE valueType get \(\)](#)  
*cast x into valueType*
- [static std::string to\\_string \(\)](#)  
*string representation of value*
- [template<typename valueRing >](#)  
[static constexpr DEVICE INLINED valueRing eval \(const valueRing &v\)](#)  
*cast x into valueRing*

### Static Public Attributes

- [static constexpr int32\\_t v = x](#)  
*actual value stored in val type*

#### 8.17.1 Detailed Description

```
template<int32\_t x>
struct aerobus::i32::val< x >
```

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

## Template Parameters

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

## 8.17.2 Member Typedef Documentation

### 8.17.2.1 enclosing\_type

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

Enclosing ring type.

### 8.17.2.2 is\_zero\_t

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

is value zero

## 8.17.3 Member Function Documentation

### 8.17.3.1 eval()

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

cast x into valueRing

## Template Parameters

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

### 8.17.3.2 get()

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

cast x into valueType

## Template Parameters

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

## 8.17.3.3 to\_string()

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

string representation of value

## 8.17.4 Member Data Documentation

## 8.17.4.1 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](#)

## 8.18 aerobus::i64::val&lt; x &gt; Struct Template Reference

values in [i64](#)

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

## Public Types

- [using inner\\_type = int32\\_t](#)  
*type of represented values*
- [using enclosing\\_type = i64](#)  
*enclosing ring type*
- [using is\\_zero\\_t = std::bool\\_constant< x==0 >](#)  
*is value zero*

## Static Public Member Functions

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

## Static Public Attributes

- `static constexpr int64_t v = x`  
*actual value*

### 8.18.1 Detailed Description

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

values in [i64](#)

#### Template Parameters

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

### 8.18.2 Member Typedef Documentation

#### 8.18.2.1 enclosing\_type

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

enclosing ring type

#### 8.18.2.2 inner\_type

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

type of represented values

#### 8.18.2.3 is\_zero\_t

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

is value zero

### 8.18.3 Member Function Documentation

#### 8.18.3.1 eval()

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

cast value in valueRing

### Template Parameters

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

#### 8.18.3.2 get()

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

cast value in valueType

### Template Parameters

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

#### 8.18.3.3 to\_string()

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

string representation

## 8.18.4 Member Data Documentation

### 8.18.4.1 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](#)

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

values (seen as types) in polynomial ring

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



## Public Types

- `using enclosing_type = polynomial< Ring >`  
*enclosing ring type*
- `using aN = coeffN`  
*heavy weight coefficient (non zero)*
- `using strip = val< coeffs... >`  
*remove largest coefficient*
- `using is_zero_t = std::bool_constant<(degree==0) &&(aN::is_zero_t::value)>`  
*true\_type if polynomial is constant zero*
- `template<size_t index>`  
`using coeff_at_t = typename coeff_at< index >::type`  
*type of coefficient at index*

## Static Public Member Functions

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

## Static Public Attributes

- `static constexpr size_t degree = sizeof...(coeffs)`  
*degree of the polynomial*
- `static constexpr bool is_zero_v = is_zero_t::value`  
*true if polynomial is constant zero*

### 8.19.1 Detailed Description

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

values (seen as types) in polynomial ring

#### Template Parameters

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

### 8.19.2 Member Typedef Documentation

#### 8.19.2.1 aN

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
```

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

heavy weight coefficient (non zero)

### 8.19.2.2 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 coeffN::
at<index>::type
```

type of coefficient at index

#### Template Parameters

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

### 8.19.2.3 enclosing\_type

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

enclosing ring type

### 8.19.2.4 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

### 8.19.2.5 strip

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

remove largest coefficient

## 8.19.3 Member Function Documentation

### 8.19.3.1 eval()

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

evaluates polynomial seen as a function operating on ValueRing

## Template Parameters

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

## Parameters

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

## Returns

$P(x)$

## 8.19.3.2 to\_string()

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

get a string representation of polynomial

## Returns

something like  $a_n X^n + \dots + a_1 X + a_0$

## 8.19.4 Member Data Documentation

## 8.19.4.1 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

## 8.19.4.2 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](#)

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

projection values in the quotient ring

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

### Public Types

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

### 8.20.1 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'
---	---------------------

### 8.20.2 Member Typedef Documentation

#### 8.20.2.1 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`

## 8.21 aerobus::zpz< p >::val< x > Struct Template Reference

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

### Public Types

- `using enclosing_type = zpz< p >`  
*enclosing ring type*
- `using is_zero_t = std::bool_constant< x% p==0 >`

## Static Public Member Functions

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

## Static Public Attributes

- `static constexpr int32_t v = x % p`  
*actual value*

## 8.21.1 Member Typedef Documentation

### 8.21.1.1 enclosing\_type

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

enclosing ring type

### 8.21.1.2 is\_zero\_t

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

## 8.21.2 Member Function Documentation

### 8.21.2.1 eval()

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

### 8.21.2.2 get()

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

### 8.21.2.3 to\_string()

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

## 8.21.3 Member Data Documentation

### 8.21.3.1 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

## 8.22 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 [enclosing\\_type](#) = [polynomial< Ring >](#)  
*enclosing ring type*
- using [aN](#) = [coeffN](#)
- using [strip](#) = [val< coeffN >](#)
- using [is\\_zero\\_t](#) = std::bool\_constant< [aN::is\\_zero\\_t::value](#) >
- template<size\_t index>  
using [coeff\\_at\\_t](#) = typename [coeff\\_at< index >](#)::type

### Static Public Member Functions

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

### Static Public Attributes

- `static constexpr size_t degree = 0`  
*degree*
- `static constexpr bool is_zero_v = is_zero_t::value`

## 8.22.1 Detailed Description

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

specialization for constants

Template Parameters

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

## 8.22.2 Member Typedef Documentation

### 8.22.2.1 aN

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

### 8.22.2.2 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
```

### 8.22.2.3 enclosing\_type

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

enclosing ring type

### 8.22.2.4 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>
```

### 8.22.2.5 strip

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

## 8.22.3 Member Function Documentation

### 8.22.3.1 eval()

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

### 8.22.3.2 to\_string()

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

## 8.22.4 Member Data Documentation

### 8.22.4.1 degree

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

degree

### 8.22.4.2 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](#)

## 8.23 aerobus::zpz< p > Struct Template Reference

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



## Classes

- struct [val](#)

## Public Types

- [using inner\\_type](#) = [int32\\_t](#)
- [template<auto x>](#)  
[using inject\\_constant\\_t](#) = [val](#)< [static\\_cast](#)< [int32\\_t](#) >(x)>
- [using zero](#) = [val](#)< 0 >
- [using one](#) = [val](#)< 1 >
- [template<typename v1 , 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
- [static constexpr bool is\\_euclidean\\_domain](#) = [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 (boolean value)*
- `template<typename v >`  
`static constexpr bool pos_v = pos_t<v>::value`  
*positivity operator (boolean value)*

### 8.23.1 Detailed Description

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

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

### 8.23.2 Member Typedef Documentation

#### 8.23.2.1 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

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

#### 8.23.2.2 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

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

#### 8.23.2.3 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

v1	a value in <a href="#">zpz::val</a>
v2	a value in <a href="#">zpz::val</a>

#### 8.23.2.4 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

v1	a value in <a href="#">zpz::val</a>
v2	a value in <a href="#">zpz::val</a>

#### 8.23.2.5 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

v1	a value in <a href="#">zpz::val</a>
v2	a value in <a href="#">zpz::val</a>

#### 8.23.2.6 inject\_constant\_t

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

#### 8.23.2.7 inner\_type

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

**8.23.2.8 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 <a href="#">zpz::val</a>
v2	a value in <a href="#">zpz::val</a>

**8.23.2.9 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 <a href="#">zpz::val</a>
v2	a value in <a href="#">zpz::val</a>

**8.23.2.10 mul\_t**

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

multiplication operator

**Template Parameters**

v1	a value in <a href="#">zpz::val</a>
v2	a value in <a href="#">zpz::val</a>

**8.23.2.11 one**

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

**8.23.2.12 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 <a href="#">zpz::val</a>
-----------	-------------------------------------

### 8.23.2.13 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 <a href="#">zpz::val</a>
<i>v2</i>	a value in <a href="#">zpz::val</a>

### 8.23.2.14 zero

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

## 8.23.3 Member Data Documentation

### 8.23.3.1 eq\_v

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

equality operator (booleanvalue)

#### Template Parameters

<i>v1</i>	a value in <a href="#">zpz::val</a>
<i>v2</i>	a value in <a href="#">zpz::val</a>

### 8.23.3.2 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 <a href="#">zpz::val</a>
v2	a value in <a href="#">zpz::val</a>

### 8.23.3.3 is\_euclidean\_domain

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

### 8.23.3.4 is\_field

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

### 8.23.3.5 lt\_v

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

strictly smaller operator (booleanvalue)

#### Template Parameters

v1	a value in <a href="#">zpz::val</a>
v2	a value in <a href="#">zpz::val</a>

### 8.23.3.6 pos\_v

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

positivity operator (boolean value)

#### Template Parameters

v1	a value in <a href="#">zpz::val</a>
----	-------------------------------------

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

- [src/aerobus.h](#)





## Chapter 9

# File Documentation

### 9.1 README.md File Reference

### 9.2 src/aerobus.h File Reference

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

Include dependency graph for aerobus.h:

### 9.3 aerobus.h

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

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

```

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

```

```

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

```

```

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

```

```

00322         val::enclosing_type::template pos_v<val>,
00323         val, typename val::enclosing_type::template sub_t<typename
    val::enclosing_type::zero, val>;
00324 } // namespace aerobus
00325
00326 namespace aerobus {
00331     template<typename Ring, typename X>
00332     requires IsRing<Ring>
00333     struct Quotient {
00336         template <typename V>
00337         struct val {
00338             public:
00339                 using type = abs_t<typename Ring::template mod_t<V, X>>;
00340         };
00341
00343         using zero = val<typename Ring::zero>;
00344
00346         using one = val<typename Ring::one>;
00347
00351         template<typename v1, typename v2>
00352         using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00353
00357         template<typename v1, typename v2>
00358         using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00359
00363         template<typename v1, typename v2>
00364         using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00365
00369         template<typename v1, typename v2>
00370         using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00371
00375         template<typename v1, typename v2>
00376         using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00377
00381         template<typename v1, typename v2>
00382         static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00383
00387         template<typename v1>
00388         using pos_t = std::true_type;
00389
00393         template<typename v>
00394         static constexpr bool pos_v = pos_t<v>::value;
00395
00397         static constexpr bool is_euclidean_domain = true;
00398
00404         template<auto x>
00405         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00406
00412         template<typename v>
00413         using inject_ring_t = val<v>;
00414     };
00415 } // namespace aerobus
00416
00417 // type_list
00418 namespace aerobus {
00420     template <typename... Ts>
00421     struct type_list;
00422
00423     namespace internal {
00424         template <typename T, typename... Us>
00425         struct pop_front_h {
00426             using tail = type_list<Us...>;
00427             using head = T;
00428         };
00429
00430         template <size_t index, typename L1, typename L2>
00431         struct split_h {
00432             private:
00433                 static_assert(index <= L2::length, "index ouf of bounds");
00434                 using a = typename L2::pop_front::type;
00435                 using b = typename L2::pop_front::tail;
00436                 using c = typename L1::template push_back<a>;
00437
00438             public:
00439                 using head = typename split_h<index - 1, c, b>::head;
00440                 using tail = typename split_h<index - 1, c, b>::tail;
00441         };
00442
00443         template <typename L1, typename L2>
00444         struct split_h<0, L1, L2> {
00445             using head = L1;
00446             using tail = L2;
00447         };
00448
00449         template <size_t index, typename L, typename T>
00450         struct insert_h {
00451             static_assert(index <= L::length, "index ouf of bounds");

```

```

00452         using s = typename L::template split<index>;
00453         using left = typename s::head;
00454         using right = typename s::tail;
00455         using ll = typename left::template push_back<T>;
00456         using type = typename ll::template concat<right>;
00457     };
00458
00459     template <size_t index, typename L>
00460     struct remove_h {
00461         using s = typename L::template split<index>;
00462         using left = typename s::head;
00463         using right = typename s::tail;
00464         using rr = typename right::pop_front::tail;
00465         using type = typename left::template concat<rr>;
00466     };
00467 } // namespace internal
00468
00472 template <typename... Ts>
00473 struct type_list {
00474     private:
00475         template <typename T>
00476         struct concat_h;
00477
00478         template <typename... Us>
00479         struct concat_h<type_list<Us...> {
00480             using type = type_list<Ts..., Us...>;
00481         };
00482
00483     public:
00484         static constexpr size_t length = sizeof...(Ts);
00485
00486         template <typename T>
00487         using push_front = type_list<T, Ts...>;
00488
00489         template <size_t index>
00490         using at = internal::type_at_t<index, Ts...>;
00491
00492         struct pop_front {
00493             using type = typename internal::pop_front_h<Ts...>::head;
00494             using tail = typename internal::pop_front_h<Ts...>::tail;
00495         };
00496
00497         template <typename T>
00498         using push_back = type_list<Ts..., T>;
00499
00500         template <typename U>
00501         using concat = typename concat_h<U>::type;
00502
00503         template <size_t index>
00504         struct split {
00505             private:
00506                 using inner = internal::split_h<index, type_list<>, type_list<Ts...>;
00507
00508             public:
00509                 using head = typename inner::head;
00510                 using tail = typename inner::tail;
00511         };
00512
00513         template <typename T, size_t index>
00514         using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00515
00516         template <size_t index>
00517         using remove = typename internal::remove_h<index, type_list<Ts...>::type;
00518     };
00519
00520     template <>
00521     struct type_list<> {
00522         static constexpr size_t length = 0;
00523
00524         template <typename T>
00525         using push_front = type_list<T>;
00526
00527         template <typename T>
00528         using push_back = type_list<T>;
00529
00530         template <typename U>
00531         using concat = U;
00532
00533         // TODO(jewave): assert index == 0
00534         template <typename T, size_t index>
00535         using insert = type_list<T>;
00536     };
00537 } // namespace aerobus
00538
00539 // i32
00540 namespace aerobus {
00541     struct i32 {

```

```

00563     using inner_type = int32_t;
00564     template<int32_t x>
00565     struct val {
00566         using enclosing_type = i32;
00567         static constexpr int32_t v = x;
00572
00573         template<typename valueType>
00574         static constexpr INLINED_DEVICE valueType get() { return static_cast<valueType>(x); }
00577
00578         using is_zero_t = std::bool_constant<x == 0>;
00580
00581         static std::string to_string() {
00582             return std::to_string(x);
00583         }
00584
00585         template<typename valueRing>
00586         static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
00587             return static_cast<valueRing>(x);
00590         }
00591     };
00592
00593     using zero = val<0>;
00594     using one = val<1>;
00595     static constexpr bool is_field = false;
00596     static constexpr bool is_euclidean_domain = true;
00600     template<auto x>
00601     using inject_constant_t = val<static_cast<int32_t>(x)>;
00602
00603     template<typename v>
00604     using inject_ring_t = v;
00605
00606 private:
00607     template<typename v1, typename v2>
00608     struct add {
00609         using type = val<v1::v + v2::v>;
00610     };
00611
00612     template<typename v1, typename v2>
00613     struct sub {
00614         using type = val<v1::v - v2::v>;
00615     };
00616
00617     template<typename v1, typename v2>
00618     struct mul {
00619         using type = val<v1::v * v2::v>;
00620     };
00621
00622     template<typename v1, typename v2>
00623     struct div {
00624         using type = val<v1::v / v2::v>;
00625     };
00626
00627     template<typename v1, typename v2>
00628     struct remainder {
00629         using type = val<v1::v % v2::v>;
00630     };
00631
00632     template<typename v1, typename v2>
00633     struct gt {
00634         using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00635     };
00636
00637     template<typename v1, typename v2>
00638     struct lt {
00639         using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00640     };
00641
00642     template<typename v1, typename v2>
00643     struct eq {
00644         using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00645     };
00646
00647     template<typename v1>
00648     struct pos {
00649         using type = std::bool_constant<(v1::v > 0)>;
00650     };
00651
00652 public:
00653     template<typename v1, typename v2>
00654     using add_t = typename add<v1, v2>::type;
00655
00656     template<typename v1, typename v2>
00657     using sub_t = typename sub<v1, v2>::type;
00658
00659     template<typename v1, typename v2>
00660     using mul_t = typename mul<v1, v2>::type;
00661

```





```

00835
00836     template<typename v1, typename v2>
00837     struct remainder {
00838         using type = val<v1::v% v2::v>;
00839     };
00840
00841     template<typename v1, typename v2>
00842     struct gt {
00843         using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00844     };
00845
00846     template<typename v1, typename v2>
00847     struct lt {
00848         using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00849     };
00850
00851     template<typename v1, typename v2>
00852     struct eq {
00853         using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00854     };
00855
00856     template<typename v>
00857     struct pos {
00858         using type = std::bool_constant<(v::v > 0)>;
00859     };
00860
00861     public:
00862     template<typename v1, typename v2>
00863     using add_t = typename add<v1, v2>::type;
00864
00865     template<typename v1, typename v2>
00866     using sub_t = typename sub<v1, v2>::type;
00867
00868     template<typename v1, typename v2>
00869     using mul_t = typename mul<v1, v2>::type;
00870
00871     template<typename v1, typename v2>
00872     using div_t = typename div<v1, v2>::type;
00873
00874     template<typename v1, typename v2>
00875     using mod_t = typename remainder<v1, v2>::type;
00876
00877     template<typename v1, typename v2>
00878     using gt_t = typename gt<v1, v2>::type;
00879
00880     template<typename v1, typename v2>
00881     static constexpr bool gt_v = gt_t<v1, v2>::value;
00882
00883     template<typename v1, typename v2>
00884     using lt_t = typename lt<v1, v2>::type;
00885
00886     template<typename v1, typename v2>
00887     static constexpr bool lt_v = lt_t<v1, v2>::value;
00888
00889     template<typename v1, typename v2>
00890     using eq_t = typename eq<v1, v2>::type;
00891
00892     template<typename v1, typename v2>
00893     static constexpr bool eq_v = eq_t<v1, v2>::value;
00894
00895     template<typename v1, typename v2>
00896     using gcd_t = gcd_t<i64, v1, v2>;
00897
00898     template<typename v>
00899     using pos_t = typename pos<v>::type;
00900
00901     template<typename v>
00902     static constexpr bool pos_v = pos_t<v>::value;
00903 };
00904 } // namespace aerobus
00905
00906 // z/pz
00907 namespace aerobus {
00908     template<int32_t p>
00909     struct zp {
00910         using inner_type = int32_t;
00911         template<int32_t x>
00912         struct val {
00913             using enclosing_type = zp<p>;
00914             static constexpr int32_t v = x % p;
00915
00916             template<typename valueType>
00917             static constexpr DEVICE INLINED valueType get() { return static_cast<valueType>(x % p); }
00918
00919             using is_zero_t = std::bool_constant<x% p == 0>;
00920             static std::string to_string() {
00921                 return std::to_string(x % p);
00922             }
00923         };
00924     };
00925 }

```



```

01105     template<typename v1, typename v2>
01106     static constexpr bool lt_v = lt_t<v1, v2>::value;
01107
01111     template<typename v1, typename v2>
01112     using eq_t = typename eq<v1, v2>::type;
01113
01117     template<typename v1, typename v2>
01118     static constexpr bool eq_v = eq_t<v1, v2>::value;
01119
01123     template<typename v1, typename v2>
01124     using gcd_t = gcd_t<i32, v1, v2>;
01125
01128     template<typename v1>
01129     using pos_t = typename pos<v1>::type;
01130
01133     template<typename v>
01134     static constexpr bool pos_v = pos_t<v>::value;
01135 };
01136 } // namespace aerobus
01137
01138 // polynomial
01139 namespace aerobus {
01140     // coeffN x^N + ...
01141     template<typename Ring>
01142     requires IsEuclideanDomain<Ring>
01143     struct polynomial {
01144         static constexpr bool is_field = false;
01145         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01146
01154         template<typename coeffN, typename... coeffs>
01155         struct val {
01157             using enclosing_type = polynomial<Ring>;
01159             static constexpr size_t degree = sizeof...(coeffs);
01161             using aN = coeffN;
01163             using strip = val<coeffs...>;
01165             using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01167             static constexpr bool is_zero_v = is_zero_t::value;
01168
01169         private:
01170             template<size_t index, typename E = void>
01171             struct coeff_at {};
01172
01173             template<size_t index>
01174             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))> {
01175                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01176             };
01177
01178             template<size_t index>
01179             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))> {
01180                 using type = typename Ring::zero;
01181             };
01182
01183         public:
01184             template<size_t index>
01185             using coeff_at_t = typename coeff_at<index>::type;
01186
01191             static std::string to_string() {
01192                 return string_helper<coeffN, coeffs...>::func();
01193             }
01194
01199             template<typename valueRing>
01200             static constexpr DEVICE INLINED valueRing eval(const valueRing& x) {
01201                 return horner_evaluation<valueRing, val>
01202                     ::template inner<0, degree + 1>
01203                     ::func(static_cast<valueRing>(0), x);
01204             }
01205         };
01206
01209         template<typename coeffN>
01210         struct val<coeffN> {
01212             using enclosing_type = polynomial<Ring>;
01214             static constexpr size_t degree = 0;
01215             using aN = coeffN;
01216             using strip = val<coeffN>;
01217             using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01218
01219             static constexpr bool is_zero_v = is_zero_t::value;
01220
01221             template<size_t index, typename E = void>
01222             struct coeff_at {};
01223
01224             template<size_t index>
01225             struct coeff_at<index, std::enable_if_t<(index == 0)> {
01226                 using type = aN;
01227             };
01228
01229             template<size_t index>

```

```

01230     struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01231         using type = typename Ring::zero;
01232     };
01233
01234     template<size_t index>
01235     using coeff_at_t = typename coeff_at<index>::type;
01236
01237     static std::string to_string() {
01238         return string_helper<coeffN>::func();
01239     }
01240
01241     template<typename valueRing>
01242     static constexpr DEVICE INLINED valueRing eval(const valueRing& x) {
01243         return static_cast<valueRing>(aN::template get<valueRing>());
01244     }
01245 };
01246
01248 using zero = val<typename Ring::zero>;
01250 using one = val<typename Ring::one>;
01252 using X = val<typename Ring::one, typename Ring::zero>;
01253
01254 private:
01255     template<typename P, typename E = void>
01256     struct simplify;
01257
01258     template<typename P1, typename P2, typename I>
01259     struct add_low;
01260
01261     template<typename P1, typename P2>
01262     struct add {
01263         using type = typename simplify<typename add_low<
01264             P1,
01265             P2,
01266             internal::make_index_sequence_reverse<
01267                 std::max(P1::degree, P2::degree) + 1
01268             >::type>::type;
01269     };
01270
01271     template<typename P1, typename P2, typename I>
01272     struct sub_low;
01273
01274     template<typename P1, typename P2, typename I>
01275     struct mul_low;
01276
01277     template<typename v1, typename v2>
01278     struct mul {
01279         using type = typename mul_low<
01280             v1,
01281             v2,
01282             internal::make_index_sequence_reverse<
01283                 v1::degree + v2::degree + 1
01284             >::type;
01285     };
01286
01287     template<typename coeff, size_t deg>
01288     struct monomial;
01289
01290     template<typename v, typename E = void>
01291     struct derive_helper {};
01292
01293     template<typename v>
01294     struct derive_helper<v, std::enable_if_t<v::degree == 0>» {
01295         using type = zero;
01296     };
01297
01298     template<typename v>
01299     struct derive_helper<v, std::enable_if_t<v::degree != 0>» {
01300         using type = typename add<
01301             typename derive_helper<typename simplify<typename v::strip>::type>::type,
01302             typename monomial<
01303                 typename Ring::template mul_t<
01304                     typename v::aN,
01305                     typename Ring::template inject_constant_t<(v::degree)>
01306                 >,
01307                 v::degree - 1
01308             >::type
01309         >::type;
01310     };
01311
01312     template<typename v1, typename v2, typename E = void>
01313     struct eq_helper {};
01314
01315     template<typename v1, typename v2>
01316     struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree>» {
01317         using type = std::false_type;
01318     };
01319

```

```

01320
01321     template<typename v1, typename v2>
01322     struct eq_helper<v1, v2, std::enable_if_t<
01323         v1::degree == v2::degree &&
01324         (v1::degree != 0 || v2::degree != 0) &&
01325         std::is_same<
01326             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01327             std::false_type
01328         >::value
01329     >
01330     > {
01331         using type = std::false_type;
01332     };
01333
01334     template<typename v1, typename v2>
01335     struct eq_helper<v1, v2, std::enable_if_t<
01336         v1::degree == v2::degree &&
01337         (v1::degree != 0 || v2::degree != 0) &&
01338         std::is_same<
01339             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01340             std::true_type
01341         >::value
01342     >> {
01343         using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01344     };
01345
01346     template<typename v1, typename v2>
01347     struct eq_helper<v1, v2, std::enable_if_t<
01348         v1::degree == v2::degree &&
01349         (v1::degree == 0)
01350     >> {
01351         using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01352     };
01353
01354     template<typename v1, typename v2, typename E = void>
01355     struct lt_helper {};
01356
01357     template<typename v1, typename v2>
01358     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)>> {
01359         using type = std::true_type;
01360     };
01361
01362     template<typename v1, typename v2>
01363     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)>> {
01364         using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01365     };
01366
01367     template<typename v1, typename v2>
01368     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)>> {
01369         using type = std::false_type;
01370     };
01371
01372     template<typename v1, typename v2, typename E = void>
01373     struct gt_helper {};
01374
01375     template<typename v1, typename v2>
01376     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)>> {
01377         using type = std::true_type;
01378     };
01379
01380     template<typename v1, typename v2>
01381     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)>> {
01382         using type = std::false_type;
01383     };
01384
01385     template<typename v1, typename v2>
01386     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)>> {
01387         using type = std::false_type;
01388     };
01389
01390     // when high power is zero : strip
01391     template<typename P>
01392     struct simplify<P, std::enable_if_t<
01393         std::is_same<
01394             typename Ring::zero,
01395             typename P::aN
01396         >::value && (P::degree > 0)
01397     >> {
01398         using type = typename simplify<typename P::strip>::type;
01399     };
01400
01401     // otherwise : do nothing
01402     template<typename P>
01403     struct simplify<P, std::enable_if_t<
01404         !std::is_same<
01405             typename Ring::zero,
01406             typename P::aN

```

```

01407         >::value && (P::degree > 0)
01408     » {
01409         using type = P;
01410     };
01411
01412     // do not simplify constants
01413     template<typename P>
01414     struct simplify<P, std::enable_if_t<P::degree == 0>» {
01415         using type = P;
01416     };
01417
01418     // addition at
01419     template<typename P1, typename P2, size_t index>
01420     struct add_at {
01421         using type =
01422             typename Ring::template add_t<
01423                 typename P1::template coeff_at_t<index>,
01424                 typename P2::template coeff_at_t<index>>;
01425     };
01426
01427     template<typename P1, typename P2, size_t index>
01428     using add_at_t = typename add_at<P1, P2, index>::type;
01429
01430     template<typename P1, typename P2, std::size_t... I>
01431     struct add_low<P1, P2, std::index_sequence<I...>» {
01432         using type = val<add_at_t<P1, P2, I>...>;
01433     };
01434
01435     // subtraction at
01436     template<typename P1, typename P2, size_t index>
01437     struct sub_at {
01438         using type =
01439             typename Ring::template sub_t<
01440                 typename P1::template coeff_at_t<index>,
01441                 typename P2::template coeff_at_t<index>>;
01442     };
01443
01444     template<typename P1, typename P2, size_t index>
01445     using sub_at_t = typename sub_at<P1, P2, index>::type;
01446
01447     template<typename P1, typename P2, std::size_t... I>
01448     struct sub_low<P1, P2, std::index_sequence<I...>» {
01449         using type = val<sub_at_t<P1, P2, I>...>;
01450     };
01451
01452     template<typename P1, typename P2>
01453     struct sub {
01454         using type = typename simplify<typename sub_low<
01455             P1,
01456             P2,
01457             internal::make_index_sequence_reverse<
01458                 std::max(P1::degree, P2::degree) + 1
01459             >::type>::type;
01460     };
01461
01462     // multiplication at
01463     template<typename v1, typename v2, size_t k, size_t index, size_t stop>
01464     struct mul_at_loop_helper {
01465         using type = typename Ring::template add_t<
01466             typename Ring::template mul_t<
01467                 typename v1::template coeff_at_t<index>,
01468                 typename v2::template coeff_at_t<k - index>
01469             >,
01470             typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
01471         >;
01472     };
01473
01474     template<typename v1, typename v2, size_t k, size_t stop>
01475     struct mul_at_loop_helper<v1, v2, k, stop, stop> {
01476         using type = typename Ring::template mul_t<
01477             typename v1::template coeff_at_t<stop>,
01478             typename v2::template coeff_at_t<0>>;
01479     };
01480
01481     template<typename v1, typename v2, size_t k, typename E = void>
01482     struct mul_at {};
01483
01484     template<typename v1, typename v2, size_t k>
01485     struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)>» {
01486         using type = typename Ring::zero;
01487     };
01488
01489     template<typename v1, typename v2, size_t k>
01490     struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)>» {
01491         using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01492     };
01493

```

```

01494     template<typename P1, typename P2, size_t index>
01495     using mul_at_t = typename mul_at<P1, P2, index>::type;
01496
01497     template<typename P1, typename P2, std::size_t... I>
01498     struct mul_low<P1, P2, std::index_sequence<I...> {
01499         using type = val<mul_at_t<P1, P2, I>...>;
01500     };
01501
01502     // division helper
01503     template< typename A, typename B, typename Q, typename R, typename E = void>
01504     struct div_helper {};
01505
01506     template<typename A, typename B, typename Q, typename R>
01507     struct div_helper<A, B, Q, R, std::enable_if_t<
01508         (R::degree < B::degree) ||
01509         (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)> {
01510         using q_type = Q;
01511         using mod_type = R;
01512         using gcd_type = B;
01513     };
01514
01515     template<typename A, typename B, typename Q, typename R>
01516     struct div_helper<A, B, Q, R, std::enable_if_t<
01517         (R::degree >= B::degree) &&
01518         !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)> {
01519     private: // NOLINT
01520         using rN = typename R::aN;
01521         using bN = typename B::aN;
01522         using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
01523             B::degree>::type;
01524         using rr = typename sub<R, typename mul<pT, B>::type>::type;
01525         using qq = typename add<Q, pT>::type;
01526     public:
01527         using q_type = typename div_helper<A, B, qq, rr>::q_type;
01528         using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
01529         using gcd_type = rr;
01530     };
01531
01532     template<typename A, typename B>
01533     struct div {
01534         static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
01535         using q_type = typename div_helper<A, B, zero, A>::q_type;
01536         using m_type = typename div_helper<A, B, zero, A>::mod_type;
01537     };
01538
01539     template<typename P>
01540     struct make_unit {
01541         using type = typename div<P, val<typename P::aN>::q_type>;
01542     };
01543
01544     template<typename coeff, size_t deg>
01545     struct monomial {
01546         using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
01547     };
01548
01549     template<typename coeff>
01550     struct monomial<coeff, 0> {
01551         using type = val<coeff>;
01552     };
01553
01554     template<typename valueRing, typename P>
01555     struct horner_evaluation {
01556         template<size_t index, size_t stop>
01557         struct inner {
01558             static constexpr DEVICE INLINED valueRing func(const valueRing& accum, const
01559 valueRing& x) {
01560                 constexpr valueRing coeff =
01561                     static_cast<valueRing>(P::template coeff_at_t<P::degree - index>::template
01562 get<valueRing>());
01563                 return horner_evaluation<valueRing, P>::template inner<index + 1, stop>::func(x *
01564 accum + coeff, x);
01565             }
01566         };
01567         template<size_t stop>
01568         struct inner<stop, stop> {
01569             static constexpr DEVICE INLINED valueRing func(const valueRing& accum, const
01570 valueRing& x) {
01571                 return accum;
01572             }
01573         };
01574     };
01575
01576     template<typename coeff, typename... coeffs>
01577     struct string_helper {
01578         static std::string func() {

```

```

01576         std::string tail = string_helper<coeffs...>::func();
01577         std::string result = "";
01578         if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
01579             return tail;
01580         } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
01581             if (sizeof...(coeffs) == 1) {
01582                 result += "x";
01583             } else {
01584                 result += "x^" + std::to_string(sizeof...(coeffs));
01585             }
01586         } else {
01587             if (sizeof...(coeffs) == 1) {
01588                 result += coeff::to_string() + " x";
01589             } else {
01590                 result += coeff::to_string()
01591                     + " x^" + std::to_string(sizeof...(coeffs));
01592             }
01593         }
01594
01595         if (!tail.empty()) {
01596             result += " + " + tail;
01597         }
01598
01599         return result;
01600     }
01601 };
01602
01603 template<typename coeff>
01604 struct string_helper<coeff> {
01605     static std::string func() {
01606         if (!std::is_same<coeff, typename Ring::zero>::value) {
01607             return coeff::to_string();
01608         } else {
01609             return "";
01610         }
01611     }
01612 };
01613
01614 public:
01615     template<typename P>
01616     using simplify_t = typename simplify<P>::type;
01617
01618     template<typename v1, typename v2>
01619     using add_t = typename add<v1, v2>::type;
01620
01621     template<typename v1, typename v2>
01622     using sub_t = typename sub<v1, v2>::type;
01623
01624     template<typename v1, typename v2>
01625     using mul_t = typename mul<v1, v2>::type;
01626
01627     template<typename v1, typename v2>
01628     using eq_t = typename eq_helper<v1, v2>::type;
01629
01630     template<typename v1, typename v2>
01631     using lt_t = typename lt_helper<v1, v2>::type;
01632
01633     template<typename v1, typename v2>
01634     using gt_t = typename gt_helper<v1, v2>::type;
01635
01636     template<typename v1, typename v2>
01637     using div_t = typename div<v1, v2>::q_type;
01638
01639     template<typename v1, typename v2>
01640     using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
01641
01642     template<typename coeff, size_t deg>
01643     using monomial_t = typename monomial<coeff, deg>::type;
01644
01645     template<typename v>
01646     using derive_t = typename derive_helper<v>::type;
01647
01648     template<typename v>
01649     using pos_t = typename Ring::template pos_t<typename v::aN>;
01650
01651     template<typename v>
01652     static constexpr bool pos_v = pos_t<v>::value;
01653
01654     template<typename v1, typename v2>
01655     using gcd_t = std::conditional_t<
01656         Ring::is_euclidean_domain,
01657         typename make_unit<gcd_t<polynomial<Ring>, v1, v2>::type,
01658         void>;
01659
01660     template<auto x>
01661     using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
01662
01663

```



```

01707     template<typename v>
01708     using inject_ring_t = val<v>;
01709 };
01710 } // namespace aerobus
01711
01712 // fraction field
01713 namespace aerobus {
01714     namespace internal {
01715         template<typename Ring, typename E = void>
01716         requires IsEuclideanDomain<Ring>
01717         struct _FractionField {};
01718
01719         template<typename Ring>
01720         requires IsEuclideanDomain<Ring>
01721         struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain> {
01722             static constexpr bool is_field = true;
01723             static constexpr bool is_euclidean_domain = true;
01724
01725         private:
01726             template<typename val1, typename val2, typename E = void>
01727             struct to_string_helper {};
01728
01729             template<typename val1, typename val2>
01730             struct to_string_helper<val1, val2,
01731                 std::enable_if_t<
01732                     Ring::template eq_t<
01733                         val2, typename Ring::one
01734                         >::value
01735                     >::value
01736                 > {
01737                 static std::string func() {
01738                     return val1::to_string();
01739                 }
01740             };
01741
01742             template<typename val1, typename val2>
01743             struct to_string_helper<val1, val2,
01744                 std::enable_if_t<
01745                     !Ring::template eq_t<
01746                         val2,
01747                         typename Ring::one
01748                         >::value
01749                     >
01750                 > {
01751                 static std::string func() {
01752                     return "(" + val1::to_string() + " ) / ( " + val2::to_string() + " )";
01753                 }
01754             };
01755         };
01756
01757     public:
01758         template<typename val1, typename val2>
01759         struct val {
01760             using x = val1;
01761             using y = val2;
01762             using is_zero_t = typename val1::is_zero_t;
01763             static constexpr bool is_zero_v = val1::is_zero_t::value;
01764
01765             using ring_type = Ring;
01766             using enclosing_type = _FractionField<Ring>;
01767
01768             static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
01769
01770             template<typename valueType>
01771             static constexpr DEVICE INLINED valueType get() {
01772                 return static_cast<valueType>(x::v) / static_cast<valueType>(y::v);
01773             }
01774
01775             static std::string to_string() {
01776                 return to_string_helper<val1, val2>::func();
01777             }
01778
01779             template<typename valueRing>
01780             static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
01781                 return x::eval(v) / y::eval(v);
01782             }
01783         };
01784
01785         using zero = val<typename Ring::zero, typename Ring::one>;
01786         using one = val<typename Ring::one, typename Ring::one>;
01787
01788         template<typename v>
01789         using inject_t = val<v, typename Ring::one>;
01790
01791         template<auto x>
01792         using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
01793         Ring::one>;
01794
01795

```

```

01821     template<typename v>
01822     using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
01823
01824     using ring_type = Ring;
01825
01826 private:
01827     template<typename v, typename E = void>
01828     struct simplify {};
01829
01830     // x = 0
01831     template<typename v>
01832     struct simplify<v, std::enable_if_t<v::x::is_zero_t::value> {
01833         using type = typename _FractionField<Ring>::zero;
01834     };
01835
01836     // x != 0
01837     template<typename v>
01838     struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value> {
01839     private:
01840         using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
01841         using newx = typename Ring::template div_t<typename v::x, _gcd>;
01842         using newy = typename Ring::template div_t<typename v::y, _gcd>;
01843
01844         using posx = std::conditional_t<
01845             !Ring::template pos_v<newx>,
01846             typename Ring::template sub_t<typename Ring::zero, newx>,
01847             newx>;
01848         using posy = std::conditional_t<
01849             !Ring::template pos_v<newy>,
01850             typename Ring::template sub_t<typename Ring::zero, newy>,
01851             newy>;
01852     public:
01853         using type = typename _FractionField<Ring>::template val<posx, posy>;
01854     };
01855
01856 public:
01857     template<typename v>
01858     using simplify_t = typename simplify<v>::type;
01859
01860 private:
01861     template<typename v1, typename v2>
01862     struct add {
01863     private:
01864         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01865         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01866         using dividend = typename Ring::template add_t<a, b>;
01867         using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01868         using g = typename Ring::template gcd_t<dividend, diviser>;
01869
01870     public:
01871         using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
01872             diviser>;
01873     };
01874
01875     template<typename v>
01876     struct pos {
01877     private:
01878         using type = std::conditional_t<
01879             (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
01880             (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
01881             std::true_type,
01882             std::false_type>;
01883     };
01884
01885     template<typename v1, typename v2>
01886     struct sub {
01887     private:
01888         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01889         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01890         using dividend = typename Ring::template sub_t<a, b>;
01891         using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01892         using g = typename Ring::template gcd_t<dividend, diviser>;
01893
01894     public:
01895         using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
01896             diviser>;
01897     };
01898
01899     template<typename v1, typename v2>
01900     struct mul {
01901     private:
01902         using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
01903         using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01904
01905     public:
01906         using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
01907     };
01908

```

```

01909     template<typename v1, typename v2, typename E = void>
01910     struct div {};
01911
01912     template<typename v1, typename v2>
01913     struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
_FractionField<Ring>::zero>::value> {
01914     private:
01915         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01916         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01917
01918     public:
01919         using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
01920     };
01921
01922     template<typename v1, typename v2>
01923     struct div<v1, v2, std::enable_if_t<
01924         std::is_same<zero, v1>::value && std::is_same<v2, zero>::value> {
01925         using type = one;
01926     };
01927
01928     template<typename v1, typename v2>
01929     struct eq {
01930         using type = std::conditional_t<
01931             std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
01932             std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
01933             std::true_type,
01934             std::false_type>;
01935     };
01936
01937     template<typename v1, typename v2, typename E = void>
01938     struct gt;
01939
01940     template<typename v1, typename v2>
01941     struct gt<v1, v2, std::enable_if_t<
01942         (eq<v1, v2>::type::value)
01943         > {
01944         using type = std::false_type;
01945     };
01946
01947     template<typename v1, typename v2>
01948     struct gt<v1, v2, std::enable_if_t<
01949         (!eq<v1, v2>::type::value) &&
01950         (!pos<v1>::type::value) && (!pos<v2>::type::value)
01951         > {
01952         using type = typename gt<
01953             typename sub<zero, v1>::type, typename sub<zero, v2>::type
01954             >::type;
01955     };
01956
01957     template<typename v1, typename v2>
01958     struct gt<v1, v2, std::enable_if_t<
01959         (!eq<v1, v2>::type::value) &&
01960         (pos<v1>::type::value) && (!pos<v2>::type::value)
01961         > {
01962         using type = std::true_type;
01963     };
01964
01965     template<typename v1, typename v2>
01966     struct gt<v1, v2, std::enable_if_t<
01967         (!eq<v1, v2>::type::value) &&
01968         (!pos<v1>::type::value) && (pos<v2>::type::value)
01969         > {
01970         using type = std::false_type;
01971     };
01972
01973     template<typename v1, typename v2>
01974     struct gt<v1, v2, std::enable_if_t<
01975         (!eq<v1, v2>::type::value) &&
01976         (pos<v1>::type::value) && (pos<v2>::type::value)
01977         > {
01978         using type = typename Ring::template gt_t<
01979             typename Ring::template mul_t<v1::x, v2::y>,
01980             typename Ring::template mul_t<v2::y, v2::x>
01981             >;
01982     };
01983
01984     public:
01985     template<typename v1, typename v2>
01986     using add_t = typename add<v1, v2>::type;
01987
01988     template<typename v1, typename v2>
01989     using mod_t = zero;
01990
01991     template<typename v1, typename v2>
01992     using gcd_t = v1;
01993
01994     template<typename v1, typename v2>

```

```

02010         using sub_t = typename sub<v1, v2>::type;
02011
02015         template<typename v1, typename v2>
02016         using mul_t = typename mul<v1, v2>::type;
02017
02021         template<typename v1, typename v2>
02022         using div_t = typename div<v1, v2>::type;
02023
02027         template<typename v1, typename v2>
02028         using eq_t = typename eq<v1, v2>::type;
02029
02033         template<typename v1, typename v2>
02034         static constexpr bool eq_v = eq<v1, v2>::type::value;
02035
02039         template<typename v1, typename v2>
02040         using gt_t = typename gt<v1, v2>::type;
02041
02045         template<typename v1, typename v2>
02046         static constexpr bool gt_v = gt<v1, v2>::type::value;
02047
02050         template<typename v1>
02051         using pos_t = typename pos<v1>::type;
02052
02055         template<typename v>
02056         static constexpr bool pos_v = pos_t<v>::value;
02057     };
02058
02059     template<typename Ring, typename E = void>
02060     requires IsEuclideanDomain<Ring>
02061     struct FractionFieldImpl {};
02062
02063     // fraction field of a field is the field itself
02064     template<typename Field>
02065     requires IsEuclideanDomain<Field>
02066     struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field> {
02067         using type = Field;
02068         template<typename v>
02069         using inject_t = v;
02070     };
02071
02072     // fraction field of a ring is the actual fraction field
02073     template<typename Ring>
02074     requires IsEuclideanDomain<Ring>
02075     struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field> {
02076         using type = _FractionField<Ring>;
02077     };
02078 } // namespace internal
02079
02083     template<typename Ring>
02084     requires IsEuclideanDomain<Ring>
02085     using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02086 } // namespace aerobus
02087
02088 // short names for common types
02089 namespace aerobus {
02092     using q32 = FractionField<i32>;
02095     using fpq32 = FractionField<polynomial<q32>>;
02098     using q64 = FractionField<i64>;
02100     using pi64 = polynomial<i64>;
02102     using pq64 = polynomial<q64>;
02104     using fpq64 = FractionField<polynomial<q64>>;
02109     template<typename Ring, typename v1, typename v2>
02110     using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02111
02115     template<int64_t p, int64_t q>
02116     using make_q64_t = typename q64::template simplify_t<
02117         typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>>;
02118
02122     template<int32_t p, int32_t q>
02123     using make_q32_t = typename q32::template simplify_t<
02124         typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>>;
02125
02130     template<typename Ring, typename v1, typename v2>
02131     using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02136     template<typename Ring, typename v1, typename v2>
02137     using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02138 } // namespace aerobus
02139
02140 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02141 namespace aerobus {
02142     namespace internal {
02143         template<typename T, size_t x, typename E = void>
02144         struct factorial {};
02145
02146         template<typename T, size_t x>
02147         struct factorial<T, x, std::enable_if_t<(x > 0)> {
02148         private:

```

```

02149         template<typename, size_t, typename>
02150         friend struct factorial;
02151     public:
02152         using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
x - 1>::type>;
02153         static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02154     };
02155
02156     template<typename T>
02157     struct factorial<T, 0> {
02158     public:
02159         using type = typename T::one;
02160         static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02161     };
02162     } // namespace internal
02163
02164     template<typename T, size_t i>
02165     using factorial_t = typename internal::factorial<T, i>::type;
02166
02167     template<typename T, size_t i>
02168     inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02169
02170     namespace internal {
02171         template<typename T, size_t k, size_t n, typename E = void>
02172         struct combination_helper {};
02173
02174         template<typename T, size_t k, size_t n>
02175         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)> {
02176             using type = typename FractionField<T>::template mul_t<
02177                 typename combination_helper<T, k - 1, n - 1>::type,
02178                 makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02179         };
02180
02181         template<typename T, size_t k, size_t n>
02182         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)> {
02183             using type = typename combination_helper<T, n - k, n>::type;
02184         };
02185
02186         template<typename T, size_t n>
02187         struct combination_helper<T, 0, n> {
02188             using type = typename FractionField<T>::one;
02189         };
02190
02191         template<typename T, size_t k, size_t n>
02192         struct combination {
02193             using type = typename internal::combination_helper<T, k, n>::type::x;
02194             static constexpr typename T::inner_type value =
02195                 internal::combination_helper<T, k, n>::type::template get<typename
T::inner_type>();
02196         };
02197     } // namespace internal
02198
02199     template<typename T, size_t k, size_t n>
02200     using combination_t = typename internal::combination<T, k, n>::type;
02201
02202     template<typename T, size_t k, size_t n>
02203     inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02204
02205     namespace internal {
02206         template<typename T, size_t m>
02207         struct bernoulli;
02208
02209         template<typename T, typename accum, size_t k, size_t m>
02210         struct bernoulli_helper {
02211             using type = typename bernoulli_helper<
02212                 T,
02213                 addfractions_t<T,
02214                     accum,
02215                     mulfractions_t<T,
02216                         makefraction_t<T,
02217                             combination_t<T, k, m + 1>,
02218                             typename T::one>,
02219                             typename bernoulli<T, k>::type
02220                         >,
02221                     k + 1,
02222                     m>::type;
02223         };
02224
02225         template<typename T, typename accum, size_t m>
02226         struct bernoulli_helper<T, accum, m, m> {
02227             using type = accum;
02228         };
02229     }
02230
02231     template<typename T, size_t m>
02232     struct bernoulli {
02233     public:
02234         using type = typename internal::bernoulli_helper<T, typename T::one, m>::type;
02235     };
02236
02237     template<typename T, size_t m>
02238     inline constexpr typename T::inner_type bernoulli_v = internal::bernoulli<T, m>::value;
02239
02240     namespace internal {
02241         template<typename T, size_t m>
02242         struct bernoulli_helper {
02243             using type = typename bernoulli_helper<T, typename T::one, m>::type;
02244         };
02245     }

```

```

02244
02245     template<typename T, size_t m>
02246     struct bernoulli {
02247         using type = typename FractionField<T>::template mul_t<
02248             typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02249             makefraction_t<T,
02250             typename T::template val<static_cast<typename T::inner_type>(-1)>,
02251             typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02252             >
02253     >;
02254
02255     template<typename floatType>
02256     static constexpr floatType value = type::template get<floatType>();
02257 };
02258
02259     template<typename T>
02260     struct bernoulli<T, 0> {
02261         using type = typename FractionField<T>::one;
02262
02263         template<typename floatType>
02264         static constexpr floatType value = type::template get<floatType>();
02265     };
02266 } // namespace internal
02267
02271     template<typename T, size_t n>
02272     using bernoulli_t = typename internal::bernoulli<T, n>::type;
02273
02278     template<typename FloatType, typename T, size_t n>
02279     inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02280
02281 // bell numbers
02282 namespace internal {
02283     template<typename T, size_t n, typename E = void>
02284     struct bell_helper;
02285
02286     template <typename T, size_t n>
02287     struct bell_helper<T, n, std::enable_if_t<(n > 1)>> {
02288         template<typename accum, size_t i, size_t stop>
02289         struct sum_helper {
02290             private:
02291                 using left = typename T::template mul_t<
02292                     combination_t<T, i, n-1>,
02293                     typename bell_helper<T, i>::type>;
02294                 using new_accum = typename T::template add_t<accum, left>;
02295             public:
02296                 using type = typename sum_helper<new_accum, i+1, stop>::type;
02297         };
02298
02299         template<typename accum, size_t stop>
02300         struct sum_helper<accum, stop, stop> {
02301             using type = accum;
02302         };
02303
02304         using type = typename sum_helper<typename T::zero, 0, n>::type;
02305     };
02306
02307     template<typename T>
02308     struct bell_helper<T, 0> {
02309         using type = typename T::one;
02310     };
02311
02312     template<typename T>
02313     struct bell_helper<T, 1> {
02314         using type = typename T::one;
02315     };
02316 } // namespace internal
02317
02321     template<typename T, size_t n>
02322     using bell_t = typename internal::bell_helper<T, n>::type;
02323
02327     template<typename T, size_t n>
02328     static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
02329
02330     namespace internal {
02331         template<typename T, int k, typename E = void>
02332         struct alternate {};
02333
02334         template<typename T, int k>
02335         struct alternate<T, k, std::enable_if_t<k % 2 == 0>> {
02336             using type = typename T::one;
02337             static constexpr typename T::inner_type value = type::template get<typename
02338 T::inner_type>();
02339         };
02340
02341         template<typename T, int k>
02342         struct alternate<T, k, std::enable_if_t<k % 2 != 0>> {
02343             using type = typename T::template sub_t<typename T::zero, typename T::one>;

```

```

02343         static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02344     };
02345     } // namespace internal
02346
02349     template<typename T, int k>
02350     using alternate_t = typename internal::alternate<T, k>::type;
02351
02352     namespace internal {
02353         template<typename T, int n, int k, typename E = void>
02354         struct stirling_helper {};
02355
02356         template<typename T>
02357         struct stirling_helper<T, 0, 0> {
02358             using type = typename T::one;
02359         };
02360
02361         template<typename T, int n>
02362         struct stirling_helper<T, n, 0, std::enable_if_t<(n > 0)>> {
02363             using type = typename T::zero;
02364         };
02365
02366         template<typename T, int n>
02367         struct stirling_helper<T, 0, n, std::enable_if_t<(n > 0)>> {
02368             using type = typename T::zero;
02369         };
02370
02371         template<typename T, int n, int k>
02372         struct stirling_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)>> {
02373             using type = typename T::template sub_t<
02374                 typename stirling_helper<T, n-1, k-1>::type,
02375                 typename T::template mul_t<
02376                     typename T::template inject_constant_t<n-1>,
02377                     typename stirling_helper<T, n-1, k>::type
02378                 >>;
02379         };
02380     } // namespace internal
02381
02386     template<typename T, int n, int k>
02387     using stirling_signed_t = typename internal::stirling_helper<T, n, k>::type;
02388
02393     template<typename T, int n, int k>
02394     using stirling_unsigned_t = abs_t<typename internal::stirling_helper<T, n, k>::type>;
02395
02400     template<typename T, int n, int k>
02401     static constexpr typename T::inner_type stirling_signed_v = stirling_signed_t<T, n, k>::v;
02402
02403
02408     template<typename T, int n, int k>
02409     static constexpr typename T::inner_type stirling_unsigned_v = stirling_unsigned_t<T, n, k>::v;
02410
02413     template<typename T, size_t k>
02414     inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
02415
02416     namespace internal {
02417         template<typename T>
02418         struct pow_scalar {
02419             template<size_t p>
02420             static constexpr DEVICE INLINED T func(const T& x) { return p == 0 ? static_cast<T>(1) :
02421                 p % 2 == 0 ? func<p/2>(x) * func<p/2>(x) :
02422                 x * func<p/2>(x) * func<p/2>(x);
02423         };
02424
02425
02426         template<typename T, typename p, size_t n, typename E = void>
02427         requires IsEuclideanDomain<T>
02428         struct pow;
02429
02430         template<typename T, typename p, size_t n>
02431         struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)>> {
02432             using type = typename T::template mul_t<
02433                 typename pow<T, p, n/2>::type,
02434                 typename pow<T, p, n/2>::type
02435             >;
02436         };
02437
02438         template<typename T, typename p, size_t n>
02439         struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)>> {
02440             using type = typename T::template mul_t<
02441                 p,
02442                 typename T::template mul_t<
02443                     typename pow<T, p, n/2>::type,
02444                     typename pow<T, p, n/2>::type
02445                 >
02446             >;
02447         };
02448

```

```

02449     template<typename T, typename p, size_t n>
02450     struct pow<T, p, n, std::enable_if_t<n == 0> { using type = typename T::one; };
02451 } // namespace internal
02452
02453 template<typename T, typename p, size_t n>
02454 using pow_t = typename internal::pow<T, p, n>::type;
02455
02456 template<typename T, typename p, size_t n>
02457 static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
02458
02459 template<typename T, size_t p>
02460 static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
02461 internal::pow_scalar<T>::template func<p>(x); }
02462
02463 namespace internal {
02464     template<typename, template<typename, size_t> typename, class>
02465     struct make_taylor_impl;
02466
02467     template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
02468     struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...> {
02469         using type = typename polynomial<FractionField<T>::template val<typename coeff_at<T,
02470 Is>::type...>;
02471     };
02472 }
02473
02474 template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
02475 using taylor = typename internal::make_taylor_impl<
02476     T,
02477     coeff_at,
02478     internal::make_index_sequence_reverse<deg + 1>::type;
02479
02480 namespace internal {
02481     template<typename T, size_t i>
02482     struct exp_coeff {
02483         using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02484     };
02485
02486     template<typename T, size_t i, typename E = void>
02487     struct sin_coeff_helper {};
02488
02489     template<typename T, size_t i>
02490     struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02491         using type = typename FractionField<T>::zero;
02492     };
02493
02494     template<typename T, size_t i>
02495     struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02496         using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02497     };
02498
02499     template<typename T, size_t i>
02500     struct sin_coeff {
02501         using type = typename sin_coeff_helper<T, i>::type;
02502     };
02503
02504     template<typename T, size_t i, typename E = void>
02505     struct sh_coeff_helper {};
02506
02507     template<typename T, size_t i>
02508     struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02509         using type = typename FractionField<T>::zero;
02510     };
02511
02512     template<typename T, size_t i>
02513     struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02514         using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02515     };
02516
02517     template<typename T, size_t i>
02518     struct sh_coeff {
02519         using type = typename sh_coeff_helper<T, i>::type;
02520     };
02521
02522     template<typename T, size_t i, typename E = void>
02523     struct cos_coeff_helper {};
02524
02525     template<typename T, size_t i>
02526     struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02527         using type = typename FractionField<T>::zero;
02528     };
02529
02530     template<typename T, size_t i>
02531     struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02532         using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02533     };
02534
02535     template<typename T, size_t i>
02536     struct cos_coeff {
02537         using type = typename cos_coeff_helper<T, i>::type;
02538     };
02539
02540     template<typename T, size_t i>
02541     struct cos_coeff {
02542         using type = typename cos_coeff_helper<T, i>::type;
02543     };
02544
02545     template<typename T, size_t i>

```



```

02546     struct cos_coeff {
02547         using type = typename cos_coeff_helper<T, i>::type;
02548     };
02549
02550     template<typename T, size_t i, typename E = void>
02551     struct cosh_coeff_helper {};
02552
02553     template<typename T, size_t i>
02554     struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02555         using type = typename FractionField<T>::zero;
02556     };
02557
02558     template<typename T, size_t i>
02559     struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02560         using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02561     };
02562
02563     template<typename T, size_t i>
02564     struct cosh_coeff {
02565         using type = typename cosh_coeff_helper<T, i>::type;
02566     };
02567
02568     template<typename T, size_t i>
02569     struct geom_coeff { using type = typename FractionField<T>::one; };
02570
02571
02572     template<typename T, size_t i, typename E = void>
02573     struct atan_coeff_helper;
02574
02575     template<typename T, size_t i>
02576     struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02577         using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;
02578     };
02579
02580     template<typename T, size_t i>
02581     struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02582         using type = typename FractionField<T>::zero;
02583     };
02584
02585     template<typename T, size_t i>
02586     struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
02587
02588     template<typename T, size_t i, typename E = void>
02589     struct asin_coeff_helper;
02590
02591     template<typename T, size_t i>
02592     struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02593         using type = makefraction_t<T,
02594             factorial_t<T, i - 1>,
02595             typename T::template mul_t<
02596                 typename T::template val<i>,
02597                 T::template mul_t<
02598                     pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
02599                     pow<T, factorial_t<T, i / 2>, 2
02600                 >
02601             >
02602         >>;
02603     };
02604
02605     template<typename T, size_t i>
02606     struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02607         using type = typename FractionField<T>::zero;
02608     };
02609
02610     template<typename T, size_t i>
02611     struct asin_coeff {
02612         using type = typename asin_coeff_helper<T, i>::type;
02613     };
02614
02615     template<typename T, size_t i>
02616     struct lnpl_coeff {
02617         using type = makefraction_t<T,
02618             alternate_t<T, i + 1>,
02619             typename T::template val<i>;
02620     };
02621
02622     template<typename T>
02623     struct lnpl_coeff<T, 0> { using type = typename FractionField<T>::zero; };
02624
02625     template<typename T, size_t i, typename E = void>
02626     struct asinh_coeff_helper;
02627
02628     template<typename T, size_t i>
02629     struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02630         using type = makefraction_t<T,
02631             typename T::template mul_t<
02632                 alternate_t<T, i / 2>,

```

```

02633         factorial_t<T, i - 1>
02634     >,
02635     typename T::template mul_t<
02636         typename T::template mul_t<
02637             typename T::template val<i>,
02638             pow_t<T, factorial_t<T, i / 2>, 2>
02639         >,
02640         pow_t<T, typename T::template inject_constant_t<4>, i / 2>
02641     >
02642 >;
02643 };
02644
02645 template<typename T, size_t i>
02646 struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02647     using type = typename FractionField<T>::zero;
02648 };
02649
02650 template<typename T, size_t i>
02651 struct asinh_coeff {
02652     using type = typename asinh_coeff_helper<T, i>::type;
02653 };
02654
02655 template<typename T, size_t i, typename E = void>
02656 struct atanh_coeff_helper;
02657
02658 template<typename T, size_t i>
02659 struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02660     // 1/i
02661     using type = typename FractionField<T>::template val<
02662         typename T::one,
02663         typename T::template inject_constant_t<i>;
02664 };
02665
02666 template<typename T, size_t i>
02667 struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02668     using type = typename FractionField<T>::zero;
02669 };
02670
02671 template<typename T, size_t i>
02672 struct atanh_coeff {
02673     using type = typename atanh_coeff_helper<T, i>::type;
02674 };
02675
02676 template<typename T, size_t i, typename E = void>
02677 struct tan_coeff_helper;
02678
02679 template<typename T, size_t i>
02680 struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0> {
02681     using type = typename FractionField<T>::zero;
02682 };
02683
02684 template<typename T, size_t i>
02685 struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0> {
02686 private:
02687     // 4^((i+1)/2)
02688     using _4p = typename FractionField<T>::template inject_t<
02689         pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2>;
02690     // 4^((i+1)/2) - 1
02691     using _4pml = typename FractionField<T>::template sub_t<_4p, typename
FractionField<T>::one>;
02692     // (-1)^((i-1)/2)
02693     using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2>;
02694     using dividend = typename FractionField<T>::template mul_t<
02695         altp,
02696         FractionField<T>::template mul_t<
02697             _4p,
02698             FractionField<T>::template mul_t<
02699                 _4pml,
02700                 bernoulli_t<T, (i + 1)>
02701             >
02702         >
02703 >;
02704 public:
02705     using type = typename FractionField<T>::template div_t<dividend,
02706         typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02707 };
02708
02709 template<typename T, size_t i>
02710 struct tan_coeff {
02711     using type = typename tan_coeff_helper<T, i>::type;
02712 };
02713
02714 template<typename T, size_t i, typename E = void>
02715 struct tanh_coeff_helper;
02716
02717 template<typename T, size_t i>
02718 struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0> {

```

```

02719         using type = typename FractionField<T>::zero;
02720     };
02721
02722     template<typename T, size_t i>
02723     struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0> {
02724     private:
02725         using _4p = typename FractionField<T>::template inject_t<
02726             pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2>;
02727         using _4pml = typename FractionField<T>::template sub_t<_4p, typename
FractionField<T>::one>;
02728         using dividend =
02729             typename FractionField<T>::template mul_t<
02730                 _4p,
02731                 typename FractionField<T>::template mul_t<
02732                     _4pml,
02733                     bernoulli_t<T, (i + 1)>::type;
02734     public:
02735         using type = typename FractionField<T>::template div_t<dividend,
02736             FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02737     };
02738
02739     template<typename T, size_t i>
02740     struct tanh_coeff {
02741         using type = typename tanh_coeff_helper<T, i>::type;
02742     };
02743 } // namespace internal
02744
02745 template<typename Integers, size_t deg>
02746 using exp = taylor<Integers, internal::exp_coeff, deg>;
02747
02748 template<typename Integers, size_t deg>
02749 using expml = typename polynomial<FractionField<Integers>>::template sub_t<
02750     exp<Integers, deg>,
02751     typename polynomial<FractionField<Integers>>::one>;
02752
02753 template<typename Integers, size_t deg>
02754 using lnpl = taylor<Integers, internal::lnpl_coeff, deg>;
02755
02756 template<typename Integers, size_t deg>
02757 using atan = taylor<Integers, internal::atan_coeff, deg>;
02758
02759 template<typename Integers, size_t deg>
02760 using sin = taylor<Integers, internal::sin_coeff, deg>;
02761
02762 template<typename Integers, size_t deg>
02763 using sinh = taylor<Integers, internal::sh_coeff, deg>;
02764
02765 template<typename Integers, size_t deg>
02766 using cosh = taylor<Integers, internal::cosh_coeff, deg>;
02767
02768 template<typename Integers, size_t deg>
02769 using cos = taylor<Integers, internal::cos_coeff, deg>;
02770
02771 template<typename Integers, size_t deg>
02772 using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
02773
02774 template<typename Integers, size_t deg>
02775 using asin = taylor<Integers, internal::asin_coeff, deg>;
02776
02777 template<typename Integers, size_t deg>
02778 using asinh = taylor<Integers, internal::asinh_coeff, deg>;
02779
02780 template<typename Integers, size_t deg>
02781 using atanh = taylor<Integers, internal::atanh_coeff, deg>;
02782
02783 template<typename Integers, size_t deg>
02784 using tan = taylor<Integers, internal::tan_coeff, deg>;
02785
02786 template<typename Integers, size_t deg>
02787 using tanh = taylor<Integers, internal::tanh_coeff, deg>;
02788 } // namespace aerobus
02789
02790 // continued fractions
02791 namespace aerobus {
02792     template<int64_t... values>
02793     struct ContinuedFraction {};
02794
02795     template<int64_t a0>
02796     struct ContinuedFraction<a0> {
02797         using type = typename q64::template inject_constant_t<a0>;
02798         static constexpr double val = static_cast<double>(a0);
02799     };
02800
02801     template<int64_t a0, int64_t... rest>
02802     struct ContinuedFraction<a0, rest...> {
02803         using type = q64::template add_t<
02804             typename q64::template inject_constant_t<a0>,

```

```

02871         typename q64::template div_t<
02872             typename q64::one,
02873             typename ContinuedFraction<rest...>::type
02874         >>;
02875         static constexpr double val = type::template get<double>();
02876     };
02877
02878     using PI_fraction =
02879     ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1>;
02880     using E_fraction =
02881     ContinuedFraction<2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1>;
02882     using SQRT2_fraction =
02883     ContinuedFraction<1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2>;
02884     using SQRT3_fraction =
02885     ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2>;
02886     // NOLINT
02887 } // namespace aerobus
02888
02889 // known polynomials
02890 namespace aerobus {
02891     // CChebyshev
02892     namespace internal {
02893         template<int kind, size_t deg, typename I>
02894         struct chebyshev_helper {
02895             using type = typename polynomial<I>::template sub_t<
02896                 typename polynomial<I>::template mul_t<
02897                     typename polynomial<I>::template mul_t<
02898                         typename polynomial<I>::template inject_constant_t<2>,
02899                         typename polynomial<I>::X>,
02900                         typename chebyshev_helper<kind, deg - 1, I>::type
02901                     >,
02902                     typename chebyshev_helper<kind, deg - 2, I>::type
02903                 >;
02904         };
02905
02906         template<typename I>
02907         struct chebyshev_helper<1, 0, I> {
02908             using type = typename polynomial<I>::one;
02909         };
02910
02911         template<typename I>
02912         struct chebyshev_helper<1, 1, I> {
02913             using type = typename polynomial<I>::X;
02914         };
02915
02916         template<typename I>
02917         struct chebyshev_helper<2, 0, I> {
02918             using type = typename polynomial<I>::one;
02919         };
02920
02921         template<typename I>
02922         struct chebyshev_helper<2, 1, I> {
02923             using type = typename polynomial<I>::template mul_t<
02924                 typename polynomial<I>::template inject_constant_t<2>,
02925                 typename polynomial<I>::X>;
02926         };
02927     } // namespace internal
02928
02929     // Laguerre
02930     namespace internal {
02931         template<size_t deg, typename I>
02932         struct laguerre_helper {
02933             using Q = FractionField<I>;
02934             using PQ = polynomial<Q>;
02935
02936         private:
02937             // Lk = (1 / k) * ((2 * k - 1 - x) * lkm1 - (k - 2)lkm2)
02938             using lnm2 = typename laguerre_helper<deg - 2, I>::type;
02939             using lnm1 = typename laguerre_helper<deg - 1, I>::type;
02940             // -x + 2k-1
02941             using p = typename PQ::template val<
02942                 typename Q::template inject_constant_t<-1>,
02943                 typename Q::template inject_constant_t<2 * deg - 1>;
02944             // 1/n
02945             using factor = typename PQ::template inject_ring_t<
02946                 typename Q::template val<typename I::one, typename I::template
02947                 inject_constant_t<deg>>;
02948
02949         public:
02950             using type = typename PQ::template mul_t <
02951                 factor,
02952                 typename PQ::template sub_t<
02953                     typename PQ::template mul_t<
02954                         p,
02955                         lnm1
02956                     >,
02957                     typename PQ::template mul_t<

```

```

02961             typename PQ::template inject_constant_t<deg-1>,
02962             lnm2
02963         >
02964     >
02965     >;
02966 };
02967
02968 template<typename I>
02969 struct laguerre_helper<0, I> {
02970     using type = typename polynomial<FractionField<I>::one>;
02971 };
02972
02973 template<typename I>
02974 struct laguerre_helper<1, I> {
02975     private:
02976         using PQ = polynomial<FractionField<I>;
02977     public:
02978         using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
02979 };
02980 } // namespace internal
02981
02982 // Bernstein
02983 namespace internal {
02984     template<size_t i, size_t m, typename I, typename E = void>
02985     struct bernstein_helper {};
02986
02987     template<typename I>
02988     struct bernstein_helper<0, 0, I> {
02989         using type = typename polynomial<I>::one;
02990     };
02991
02992     template<size_t i, size_t m, typename I>
02993     struct bernstein_helper<i, m, I, std::enable_if_t<
02994         (m > 0) && (i == 0)>> {
02995     private:
02996         using P = polynomial<I>;
02997     public:
02998         using type = typename P::template mul_t<
02999             typename P::template sub_t<typename P::one, typename P::X>,
03000             typename bernstein_helper<i, m-1, I>::type>;
03001     };
03002
03003     template<size_t i, size_t m, typename I>
03004     struct bernstein_helper<i, m, I, std::enable_if_t<
03005         (m > 0) && (i == m)>> {
03006     private:
03007         using P = polynomial<I>;
03008     public:
03009         using type = typename P::template mul_t<
03010             typename P::X,
03011             typename bernstein_helper<i-1, m-1, I>::type>;
03012     };
03013
03014     template<size_t i, size_t m, typename I>
03015     struct bernstein_helper<i, m, I, std::enable_if_t<
03016         (m > 0) && (i > 0) && (i < m)>> {
03017     private:
03018         using P = polynomial<I>;
03019     public:
03020         using type = typename P::template add_t<
03021             typename P::template mul_t<
03022                 typename P::template sub_t<typename P::one, typename P::X>,
03023                 typename bernstein_helper<i, m-1, I>::type>,
03024             typename P::template mul_t<
03025                 typename P::X,
03026                 typename bernstein_helper<i-1, m-1, I>::type>;
03027     };
03028 } // namespace internal
03029
03030 namespace known_polynomials {
03031     enum hermite_kind {
03032         probabilist,
03033         physicist
03034     };
03035 }
03036
03037 // hermite
03038 namespace internal {
03039     template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03040     struct hermite_helper {};
03041
03042     template<size_t deg, typename I>
03043     struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
03044     private:
03045         using hnm1 = typename hermite_helper<deg - 1,
03046             known_polynomials::hermite_kind::probabilist, I>::type;
03047         using hnm2 = typename hermite_helper<deg - 2,

```

```

known_polynomials::hermite_kind::probabilist, I>::type;
03050
03051     public:
03052         using type = typename polynomial<I>::template sub_t<
03053             typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnm1>,
03054             typename polynomial<I>::template mul_t<
03055                 typename polynomial<I>::template inject_constant_t<deg - 1>,
03056                 hnm2
03057             >
03058         >;
03059     };
03060
03061     template<size_t deg, typename I>
03062     struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03063     private:
03064         using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
I>::type;
03065         using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
I>::type;
03066
03067     public:
03068         using type = typename polynomial<I>::template sub_t<
03069             // 2X Hn-1
03070             typename polynomial<I>::template mul_t<
03071                 typename pi64::val<typename I::template inject_constant_t<2>,
03072                 typename I::zero>, hnm1>,
03073
03074             typename polynomial<I>::template mul_t<
03075                 typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03076                 hnm2
03077             >
03078         >;
03079     };
03080
03081     template<typename I>
03082     struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03083     public:
03084         using type = typename polynomial<I>::one;
03085     };
03086
03087     template<typename I>
03088     struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03089     public:
03090         using type = typename polynomial<I>::X;
03091     };
03092
03093     template<typename I>
03094     struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03095     public:
03096         using type = typename pi64::one;
03097     };
03098
03099     template<typename I>
03100     struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03101     public:
03102         using type = typename polynomial<I>::template val<
03103             typename I::template inject_constant_t<2>,
03104             typename I::zero>;
03105     };
03106 } // namespace internal
03107
03108 // legendre
03109 namespace internal {
03110     template<size_t n, typename I>
03111     struct legendre_helper {
03112     private:
03113         using Q = FractionField<I>;
03114         using PQ = polynomial<Q>;
03115         // 1/n constant
03116         // (2n-1)/n X
03117         using fact_left = typename PQ::template monomial_t<
03118             makefraction_t<I,
03119                 typename I::template inject_constant_t<2*n-1>,
03120                 typename I::template inject_constant_t<n>
03121             >,
03122             1>;
03123         // (n-1) / n
03124         using fact_right = typename PQ::template val<
03125             makefraction_t<I,
03126                 typename I::template inject_constant_t<n-1>,
03127                 typename I::template inject_constant_t<n>>;
03128     public:
03129         using type = PQ::template sub_t<
03130             typename PQ::template mul_t<
03131                 fact_left,
03132                 typename legendre_helper<n-1, I>::type
03133             >,
03134             typename PQ::template mul_t<
03135                 fact_right,

```

```

03134         typename legendre_helper<n-2, I>::type
03135     >
03136     >;
03137 };
03138
03139 template<typename I>
03140 struct legendre_helper<0, I> {
03141     using type = typename polynomial<FractionField<I>::one;
03142 };
03143
03144 template<typename I>
03145 struct legendre_helper<1, I> {
03146     using type = typename polynomial<FractionField<I>::X;
03147 };
03148 } // namespace internal
03149
03150 // bernoulli polynomials
03151 namespace internal {
03152     template<size_t n>
03153     struct bernoulli_coeff {
03154         template<typename T, size_t i>
03155         struct inner {
03156             private:
03157                 using F = FractionField<T>;
03158             public:
03159                 using type = typename F::template mul_t<
03160                     typename F::template inject_ring_t<combination_t<T, i, n>,
03161                     bernoulli_t<T, n-i>
03162                 >;
03163         };
03164     };
03165 } // namespace internal
03166
03167 namespace known_polynomials {
03176     template<size_t deg, typename I = aerobus::i64>
03177     using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
03178
03179     template<size_t deg, typename I = aerobus::i64>
03180     using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
03181
03182     template<size_t deg, typename I = aerobus::i64>
03183     using laguerre = typename internal::laguerre_helper<deg, I>::type;
03184
03185     template<size_t deg, typename I = aerobus::i64>
03186     using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
03187 I>::type;
03188
03189     template<size_t deg, typename I = aerobus::i64>
03190     using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
03191
03192     template<size_t i, size_t m, typename I = aerobus::i64>
03193     using bernstein = typename internal::bernstein_helper<i, m, I>::type;
03194
03195     template<size_t deg, typename I = aerobus::i64>
03196     using legendre = typename internal::legendre_helper<deg, I>::type;
03197
03198     template<size_t deg, typename I = aerobus::i64>
03199     using bernoulli = Taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
03200 } // namespace known_polynomials
03201 } // namespace aerobus
03202
03203 #ifdef AEROBUS_CONWAY_IMPORTS
03204 // conway polynomials
03205 namespace aerobus {
03206     template<int p, int n>
03207     struct ConwayPolynomial {};
03208 }
03209
03210 #ifndef DO_NOT_DOCUMENT
03211 #define ZPV ZPZ::template val
03212 #define POLYV aerobus::polynomial<ZPV>::template val
03213
03214 template<> struct ConwayPolynomial<2, 1> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<1>; }; // NOLINT
03215
03216 template<> struct ConwayPolynomial<2, 2> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<1>, ZPV<1>; }; // NOLINT
03217
03218 template<> struct ConwayPolynomial<2, 3> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<1>, ZPV<1>; }; // NOLINT
03219
03220 template<> struct ConwayPolynomial<2, 4> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<1>; }; // NOLINT
03221
03222 template<> struct ConwayPolynomial<2, 5> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<0>, ZPV<1>; }; // NOLINT
03223
03224 template<> struct ConwayPolynomial<2, 6> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<1>, ZPV<1>, ZPV<0>, ZPV<1>, ZPV<1>; }; // NOLINT
03225
03226 template<> struct ConwayPolynomial<2, 7> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<1>, ZPV<1>; }; // NOLINT
03227
03228 template<> struct ConwayPolynomial<2, 8> { using ZPV = aerobus::zpv<2>; using type =

```





Generated by Doxygen



Generated by Doxygen

Generated by Doxygen

Generated by Doxygen





Generated by Doxygen





Generated by Doxygen



Generated by Doxygen



[illegible]





Generated by Doxygen





Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen





[illegible]

Generated by Doxygen



Generated by Doxygen



Generated by Doxygen

Generated by Doxygen

Generated by Doxygen





[illegible]

Generated by Doxygen



Generated by Doxygen

Generated by Doxygen

Generated by Doxygen





Generated by Doxygen

Generated by Doxygen

Generated by Doxygen





Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen



Generated by Doxygen

Generated by Doxygen

Generated by Doxygen





Generated by Doxygen



Generated by Doxygen

[illegible]

[illegible]



## Chapter 10

# Examples

### 10.1 QuotientRing

inject a 'constant' in quotient ring

inject a 'constant' in quotient ring<i32, i32::val<2>>::inject\_constant\_t<1>>

Template Parameters

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

### 10.2 type\_list

A list of types <int, double, float>

A list of types <int, double, float>

Template Parameters

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

### 10.3 i32::template

inject a native constant

inject a native constant

Template Parameters

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



## 10.4 i32::add\_t

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

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

Template Parameters

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

## 10.5 i32::sub\_t

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

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

Template Parameters

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

## 10.6 i32::mul\_t

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

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

Template Parameters

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

## 10.7 i32::div\_t

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

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

Template Parameters

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



## 10.8 i32::gt\_t

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

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

### Template Parameters

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

## 10.9 i32::eq\_t

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

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

### Template Parameters

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

## 10.10 i32::eq\_v

equality operator (boolean value)

equality operator (boolean value)

### Template Parameters

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

## 10.11 i32::gcd\_t

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

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

### Template Parameters

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

## 10.12 i32::pos\_t

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

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

Template Parameters

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

## 10.13 i32::pos\_v

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

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

Template Parameters

$v$	a value in i32 <code>&lt;i32::val&lt;1&gt;&gt;</code>
-----	---

## 10.14 i64::template

injects constant as an i64 value

injects constant as an i64 value

Template Parameters

$x$	<code>inject_constant_t&lt;2&gt;</code>
-----	---

## 10.15 i64::add\_t

addition operator

addition operator

Template Parameters

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

## 10.16 i64::sub\_t

subtraction operator

subtraction operator

Template Parameters

<i>v1</i>	: an element of <a href="#">aerobus::i64::val</a>
<i>v2</i>	: an element of <a href="#">aerobus::i64::val</a> <i64::val<1>, i64::val<2>>

## 10.17 i64::mul\_t

multiplication operator

multiplication operator

Template Parameters

<i>v1</i>	: an element of <a href="#">aerobus::i64::val</a>
<i>v2</i>	: an element of <a href="#">aerobus::i64::val</a> <i64::val<1>, i64::val<2>>

## 10.18 i64::div\_t

division operator integer division

division operator integer division

Template Parameters

<i>v1</i>	: an element of <a href="#">aerobus::i64::val</a>
<i>v2</i>	: an element of <a href="#">aerobus::i64::val</a> <i64::val<1>, i64::val<2>>

## 10.19 i64::mod\_t

modulus operator

modulus operator

Template Parameters

<i>v1</i>	: an element of <a href="#">aerobus::i64::val</a>
<i>v2</i>	: an element of <a href="#">aerobus::i64::val</a> <i64::val<6>, i64::val<15>>

## 10.20 i64::gt\_t

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

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

### Template Parameters

<code>v1</code>	: an element of <a href="#">aerobus::i64::val</a>
<code>v2</code>	: an element of <a href="#">aerobus::i64::val</a> <code>&lt;i64::val&lt;2&gt;, i64::val&lt;1&gt;&gt;</code>

## 10.21 i64::lt\_t

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

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

### Template Parameters

<code>v1</code>	: an element of <a href="#">aerobus::i64::val</a>
<code>v2</code>	: an element of <a href="#">aerobus::i64::val</a> <code>&lt;i64::val&lt;1&gt;, i64::val&lt;2&gt;&gt;</code>

## 10.22 i64::lt\_v

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

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

### Template Parameters

<code>v1</code>	: an element of <a href="#">aerobus::i64::val</a>
<code>v2</code>	: an element of <a href="#">aerobus::i64::val</a> <code>&lt;i64::val&lt;1&gt;, i64::val&lt;2&gt;&gt;</code>

## 10.23 i64::eq\_t

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

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

### Template Parameters

<code>v1</code>	: an element of <a href="#">aerobus::i64::val</a>
<code>v2</code>	: an element of <a href="#">aerobus::i64::val</a> <code>&lt;i64::val&lt;2&gt;, i64::val&lt;2&gt;&gt;</code>

## 10.24 i64::eq\_v

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

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

Template Parameters

<code>v1</code>	: an element of <a href="#">aerobus::i64::val</a>
<code>v2</code>	: an element of <a href="#">aerobus::i64::val</a> <code>&lt;i64::val&lt;2&gt;, i64::val&lt;2&gt;&gt;</code>

## 10.25 i64::gcd\_t

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

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

Template Parameters

<code>v1</code>	: an element of <a href="#">aerobus::i64::val</a>
<code>v2</code>	: an element of <a href="#">aerobus::i64::val</a> <code>&lt;i64::val&lt;6&gt;, i64::val&lt;15&gt;&gt;</code>

## 10.26 i64::pos\_t

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

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

Template Parameters

<code>v1</code>	: an element of <a href="#">aerobus::i64::val</a> <code>&lt;i64::val&lt;1&gt;&gt;</code>
-----------------	--

## 10.27 i64::pos\_v

positivity yields  $v > 0$  as boolean value

positivity yields  $v > 0$  as boolean value

Template Parameters

<code>v</code>	: an element of <a href="#">aerobus::i64::val</a> <code>&lt;i64::val&lt;1&gt;&gt;</code>
----------------	--

## 10.28 polynomial

makes the constant (native type) polynomial `a_0`

makes the constant (native type) polynomial `a_0`

Template Parameters

<code>x</code>	<code>&lt;i32&gt;::template inject_constant_t&lt;2&gt;</code>
----------------	---

## 10.29 q32::add\_t

addition operator

addition operator

Template Parameters

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

## 10.30 FractionField

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

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

Template Parameters

<code>Ring</code>	<code>&lt;i64&gt;</code> is q64 (rationals with 64 bits numerator and denominator)
-------------------	--

## 10.31 aerobus::ContinuedFraction

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

represents a continued fraction  $a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots}}$  [ [https://en.wikipedia.org/wiki/Continued\\_fraction](https://en.wikipedia.org/wiki/Continued_fraction)](See in Wikipedia)

Template Parameters

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

$\langle 1, 1, 1 \rangle$  represents  $1 + \frac{1}{1}$

## 10.32 PI\_fraction::val

representation of  $\pi$  as a continued fraction -> 3.14...

## 10.33 E\_fraction::val

approximation of  $e$  -> 2.718...

approximation of  $e$  -> 2.718...





# Index

abs\_t  
  aerobus, 21

add\_t  
  aerobus::i32, 48  
  aerobus::i64, 51  
  aerobus::polynomial< Ring >, 57  
  aerobus::Quotient< Ring, X >, 64  
  aerobus::zpz< p >, 86

addfractions\_t  
  aerobus, 21

aerobus, 17  
  abs\_t, 21  
  addfractions\_t, 21  
  aligned\_malloc, 31  
  alternate\_t, 21  
  alternate\_v, 31  
  asin, 21  
  asinh, 22  
  atan, 22  
  atanh, 22  
  bell\_t, 22  
  bernoulli\_t, 23  
  bernoulli\_v, 32  
  combination\_t, 23  
  combination\_v, 32  
  cos, 23  
  cosh, 24  
  E\_fraction, 24  
  exp, 24  
  expm1, 24  
  factorial\_t, 24  
  factorial\_v, 32  
  field, 31  
  fpq32, 25  
  fpq64, 25  
  FractionField, 25  
  gcd\_t, 25  
  geometric\_sum, 25  
  lnp1, 26  
  make\_q32\_t, 26  
  make\_q64\_t, 26  
  makefraction\_t, 26  
  mulfractions\_t, 27  
  pi64, 27  
  PI\_fraction, 27  
  pow\_t, 27  
  pq64, 27  
  q32, 28  
  q64, 28  
  sin, 28  
  sinh, 28  
  SQRT2\_fraction, 28  
  SQRT3\_fraction, 29  
  stirling\_signed\_t, 29  
  stirling\_unsigned\_t, 29  
  tan, 29  
  tanh, 30  
  taylor, 30  
  vadd\_t, 30  
  vmul\_t, 30

aerobus::ContinuedFraction< a0 >, 44  
  type, 45  
  val, 45

aerobus::ContinuedFraction< a0, rest... >, 45  
  type, 46  
  val, 46

aerobus::ContinuedFraction< values >, 44

aerobus::ConwayPolynomial, 46

aerobus::i32, 46  
  add\_t, 48  
  div\_t, 48  
  eq\_t, 48  
  eq\_v, 49  
  gcd\_t, 48  
  gt\_t, 48  
  inject\_constant\_t, 48  
  inject\_ring\_t, 48  
  inner\_type, 48  
  is\_euclidean\_domain, 49  
  is\_field, 49  
  lt\_t, 48  
  mod\_t, 48  
  mul\_t, 49  
  one, 49  
  pos\_t, 49  
  pos\_v, 50  
  sub\_t, 49  
  zero, 49

aerobus::i32::val< x >, 72  
  enclosing\_type, 73  
  eval, 73  
  get, 73  
  is\_zero\_t, 73  
  to\_string, 74  
  v, 74

aerobus::i64, 50  
  add\_t, 51  
  div\_t, 51

- eq\_t, 51
- eq\_v, 53
- gcd\_t, 51
- gt\_t, 52
- gt\_v, 53
- inject\_constant\_t, 52
- inject\_ring\_t, 52
- inner\_type, 52
- is\_euclidean\_domain, 54
- is\_field, 54
- lt\_t, 52
- lt\_v, 54
- mod\_t, 52
- mul\_t, 52
- one, 53
- pos\_t, 53
- pos\_v, 54
- sub\_t, 53
- zero, 53
- aerobus::i64::val< x >, 74
  - enclosing\_type, 75
  - eval, 75
  - get, 76
  - inner\_type, 75
  - is\_zero\_t, 75
  - to\_string, 76
  - v, 76
- aerobus::internal, 33
  - index\_sequence\_reverse, 36
  - is\_instantiation\_of\_v, 36
  - make\_index\_sequence\_reverse, 36
  - type\_at\_t, 36
- aerobus::is\_prime< n >, 54
  - value, 55
- aerobus::IsEuclideanDomain, 41
- aerobus::IsField, 41
- aerobus::IsRing, 42
- aerobus::known\_polynomials, 36
  - bernoulli, 37
  - bernstein, 38
  - chebyshev\_T, 38
  - chebyshev\_U, 38
  - hermite\_kind, 40
  - hermite\_phys, 39
  - hermite\_prob, 39
  - laguerre, 39
  - legendre, 40
  - physicist, 40
  - probabilist, 40
- aerobus::polynomial< Ring >, 55
  - add\_t, 57
  - derive\_t, 57
  - div\_t, 57
  - eq\_t, 57
  - gcd\_t, 58
  - gt\_t, 58
  - inject\_constant\_t, 58
  - inject\_ring\_t, 58
  - is\_euclidean\_domain, 61
  - is\_field, 61
  - lt\_t, 58
  - mod\_t, 59
  - monomial\_t, 59
  - mul\_t, 59
  - one, 60
  - pos\_t, 60
  - pos\_v, 61
  - simplify\_t, 60
  - sub\_t, 60
  - X, 61
  - zero, 61
- aerobus::polynomial< Ring >::val< coeffN >, 82
  - aN, 83
  - coeff\_at\_t, 83
  - degree, 84
  - enclosing\_type, 83
  - eval, 84
  - is\_zero\_t, 83
  - is\_zero\_v, 84
  - strip, 83
  - to\_string, 84
- aerobus::polynomial< Ring >::val< coeffN >::coeff\_at< index, E >, 43
- aerobus::polynomial< Ring >::val< coeffN >::coeff\_at< index, std::enable\_if\_t<(index< 0 || index > 0)> >, 43
  - type, 43
- aerobus::polynomial< Ring >::val< coeffN >::coeff\_at< index, std::enable\_if\_t<(index==0)> >, 44
  - type, 44
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 76
  - aN, 77
  - coeff\_at\_t, 78
  - degree, 79
  - enclosing\_type, 78
  - eval, 78
  - is\_zero\_t, 78
  - is\_zero\_v, 79
  - strip, 78
  - to\_string, 79
- aerobus::Quotient< Ring, X >, 62
  - add\_t, 64
  - div\_t, 64
  - eq\_t, 64
  - eq\_v, 66
  - inject\_constant\_t, 64
  - inject\_ring\_t, 65
  - is\_euclidean\_domain, 66
  - mod\_t, 65
  - mul\_t, 65
  - one, 65
  - pos\_t, 65
  - pos\_v, 66
  - zero, 66
- aerobus::Quotient< Ring, X >::val< V >, 80

- type, 80
- aerobus::type\_list< Ts >, 68
  - at, 69
  - concat, 69
  - insert, 69
  - length, 70
  - push\_back, 69
  - push\_front, 70
  - remove, 70
- aerobus::type\_list< Ts >::pop\_front, 61
  - tail, 62
  - type, 62
- aerobus::type\_list< Ts >::split< index >, 67
  - head, 67
  - tail, 67
- aerobus::type\_list<>, 70
  - concat, 71
  - insert, 71
  - length, 72
  - push\_back, 71
  - push\_front, 71
- aerobus::zpz< p >, 84
  - add\_t, 86
  - div\_t, 86
  - eq\_t, 86
  - eq\_v, 89
  - gcd\_t, 87
  - gt\_t, 87
  - gt\_v, 89
  - inject\_constant\_t, 87
  - inner\_type, 87
  - is\_euclidean\_domain, 90
  - is\_field, 90
  - lt\_t, 87
  - lt\_v, 90
  - mod\_t, 88
  - mul\_t, 88
  - one, 88
  - pos\_t, 88
  - pos\_v, 90
  - sub\_t, 89
  - zero, 89
- aerobus::zpz< p >::val< x >, 80
  - enclosing\_type, 81
  - eval, 81
  - get, 81
  - is\_zero\_t, 81
  - to\_string, 81
  - v, 82
- aligned\_malloc
  - aerobus, 31
- alternate\_t
  - aerobus, 21
- alternate\_v
  - aerobus, 31
- aN
  - aerobus::polynomial< Ring >::val< coeffN >, 83
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 77
- asin
  - aerobus, 21
- asinh
  - aerobus, 22
- at
  - aerobus::type\_list< Ts >, 69
- atan
  - aerobus, 22
- atanh
  - aerobus, 22
- bell\_t
  - aerobus, 22
- bernoulli
  - aerobus::known\_polynomials, 37
- bernoulli\_t
  - aerobus, 23
- bernoulli\_v
  - aerobus, 32
- bernstein
  - aerobus::known\_polynomials, 38
- chebyshev\_T
  - aerobus::known\_polynomials, 38
- chebyshev\_U
  - aerobus::known\_polynomials, 38
- coeff\_at\_t
  - aerobus::polynomial< Ring >::val< coeffN >, 83
  - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 78
- combination\_t
  - aerobus, 23
- combination\_v
  - aerobus, 32
- concat
  - aerobus::type\_list< Ts >, 69
  - aerobus::type\_list<>, 71
- cos
  - aerobus, 23
- cosh
  - aerobus, 24
- degree
  - aerobus::polynomial< Ring >::val< coeffN >, 84
  - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 79
- derive\_t
  - aerobus::polynomial< Ring >, 57
- div\_t
  - aerobus::i32, 48
  - aerobus::i64, 51
  - aerobus::polynomial< Ring >, 57
  - aerobus::Quotient< Ring, X >, 64
  - aerobus::zpz< p >, 86
- E\_fraction
  - aerobus, 24

- enclosing\_type
  - aerobus::i32::val< x >, 73
  - aerobus::i64::val< x >, 75
  - aerobus::polynomial< Ring >::val< coeffN >, 83
  - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 78
  - aerobus::zpz< p >::val< x >, 81
- eq\_t
  - aerobus::i32, 48
  - aerobus::i64, 51
  - aerobus::polynomial< Ring >, 57
  - aerobus::Quotient< Ring, X >, 64
  - aerobus::zpz< p >, 86
- eq\_v
  - aerobus::i32, 49
  - aerobus::i64, 53
  - aerobus::Quotient< Ring, X >, 66
  - aerobus::zpz< p >, 89
- eval
  - aerobus::i32::val< x >, 73
  - aerobus::i64::val< x >, 75
  - aerobus::polynomial< Ring >::val< coeffN >, 84
  - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 78
  - aerobus::zpz< p >::val< x >, 81
- exp
  - aerobus, 24
- expm1
  - aerobus, 24
- factorial\_t
  - aerobus, 24
- factorial\_v
  - aerobus, 32
- field
  - aerobus, 31
- fpq32
  - aerobus, 25
- fpq64
  - aerobus, 25
- FractionField
  - aerobus, 25
- gcd\_t
  - aerobus, 25
  - aerobus::i32, 48
  - aerobus::i64, 51
  - aerobus::polynomial< Ring >, 58
  - aerobus::zpz< p >, 87
- geometric\_sum
  - aerobus, 25
- get
  - aerobus::i32::val< x >, 73
  - aerobus::i64::val< x >, 76
  - aerobus::zpz< p >::val< x >, 81
- gt\_t
  - aerobus::i32, 48
  - aerobus::i64, 52
  - aerobus::polynomial< Ring >, 58
- aerobus::zpz< p >, 87
  - gt\_v
    - aerobus::i64, 53
    - aerobus::zpz< p >, 89
  - head
    - aerobus::type\_list< Ts >::split< index >, 67
  - hermite\_kind
    - aerobus::known\_polynomials, 40
  - hermite\_phys
    - aerobus::known\_polynomials, 39
  - hermite\_prob
    - aerobus::known\_polynomials, 39
  - index\_sequence\_reverse
    - aerobus::internal, 36
  - inject\_constant\_t
    - aerobus::i32, 48
    - aerobus::i64, 52
    - aerobus::polynomial< Ring >, 58
    - aerobus::Quotient< Ring, X >, 64
    - aerobus::zpz< p >, 87
  - inject\_ring\_t
    - aerobus::i32, 48
    - aerobus::i64, 52
    - aerobus::polynomial< Ring >, 58
    - aerobus::Quotient< Ring, X >, 65
  - inner\_type
    - aerobus::i32, 48
    - aerobus::i64, 52
    - aerobus::i64::val< x >, 75
    - aerobus::zpz< p >, 87
  - insert
    - aerobus::type\_list< Ts >, 69
    - aerobus::type\_list<>, 71
  - Introduction, 1
  - is\_euclidean\_domain
    - aerobus::i32, 49
    - aerobus::i64, 54
    - aerobus::polynomial< Ring >, 61
    - aerobus::Quotient< Ring, X >, 66
    - aerobus::zpz< p >, 90
  - is\_field
    - aerobus::i32, 49
    - aerobus::i64, 54
    - aerobus::polynomial< Ring >, 61
    - aerobus::zpz< p >, 90
  - is\_instantiation\_of\_v
    - aerobus::internal, 36
  - is\_zero\_t
    - aerobus::i32::val< x >, 73
    - aerobus::i64::val< x >, 75
    - aerobus::polynomial< Ring >::val< coeffN >, 83
    - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 78
    - aerobus::zpz< p >::val< x >, 81
  - is\_zero\_v
    - aerobus::polynomial< Ring >::val< coeffN >, 84

- aerobus::polynomial< Ring >::val< coeffN, coeffs  
>, 79
- laguerre
  - aerobus::known\_polynomials, 39
- legendre
  - aerobus::known\_polynomials, 40
- length
  - aerobus::type\_list< Ts >, 70
  - aerobus::type\_list<>, 72
- lnp1
  - aerobus, 26
- lt\_t
  - aerobus::i32, 48
  - aerobus::i64, 52
  - aerobus::polynomial< Ring >, 58
  - aerobus::zpz< p >, 87
- lt\_v
  - aerobus::i64, 54
  - aerobus::zpz< p >, 90
- make\_index\_sequence\_reverse
  - aerobus::internal, 36
- make\_q32\_t
  - aerobus, 26
- make\_q64\_t
  - aerobus, 26
- makefraction\_t
  - aerobus, 26
- mod\_t
  - aerobus::i32, 48
  - aerobus::i64, 52
  - aerobus::polynomial< Ring >, 59
  - aerobus::Quotient< Ring, X >, 65
  - aerobus::zpz< p >, 88
- monomial\_t
  - aerobus::polynomial< Ring >, 59
- mul\_t
  - aerobus::i32, 49
  - aerobus::i64, 52
  - aerobus::polynomial< Ring >, 59
  - aerobus::Quotient< Ring, X >, 65
  - aerobus::zpz< p >, 88
- mulfractions\_t
  - aerobus, 27
- one
  - aerobus::i32, 49
  - aerobus::i64, 53
  - aerobus::polynomial< Ring >, 60
  - aerobus::Quotient< Ring, X >, 65
  - aerobus::zpz< p >, 88
- physicist
  - aerobus::known\_polynomials, 40
- pi64
  - aerobus, 27
- PI\_fraction
  - aerobus, 27
- pos\_t
  - aerobus::i32, 49
  - aerobus::i64, 53
  - aerobus::polynomial< Ring >, 60
  - aerobus::Quotient< Ring, X >, 65
  - aerobus::zpz< p >, 88
- pos\_v
  - aerobus::i32, 50
  - aerobus::i64, 54
  - aerobus::polynomial< Ring >, 61
  - aerobus::Quotient< Ring, X >, 66
  - aerobus::zpz< p >, 90
- pow\_t
  - aerobus, 27
- pq64
  - aerobus, 27
- probabilist
  - aerobus::known\_polynomials, 40
- push\_back
  - aerobus::type\_list< Ts >, 69
  - aerobus::type\_list<>, 71
- push\_front
  - aerobus::type\_list< Ts >, 70
  - aerobus::type\_list<>, 71
- q32
  - aerobus, 28
- q64
  - aerobus, 28
- README.md, 93
- remove
  - aerobus::type\_list< Ts >, 70
- simplify\_t
  - aerobus::polynomial< Ring >, 60
- sin
  - aerobus, 28
- sinh
  - aerobus, 28
- SQRT2\_fraction
  - aerobus, 28
- SQRT3\_fraction
  - aerobus, 29
- src/aerobus.h, 93
- stirling\_signed\_t
  - aerobus, 29
- stirling\_unsigned\_t
  - aerobus, 29
- strip
  - aerobus::polynomial< Ring >::val< coeffN >, 83
  - aerobus::polynomial< Ring >::val< coeffN, coeffs  
>, 78
- sub\_t
  - aerobus::i32, 49
  - aerobus::i64, 53
  - aerobus::polynomial< Ring >, 60
  - aerobus::zpz< p >, 89

tail  
     aerobus::type\_list< Ts >::pop\_front, 62  
     aerobus::type\_list< Ts >::split< index >, 67  
 tan  
     aerobus, 29  
 tanh  
     aerobus, 30  
 taylor  
     aerobus, 30  
 to\_string  
     aerobus::i32::val< x >, 74  
     aerobus::i64::val< x >, 76  
     aerobus::polynomial< Ring >::val< coeffN >, 84  
     aerobus::polynomial< Ring >::val< coeffN, coeffs  
         >, 79  
     aerobus::zpz< p >::val< x >, 81  
 type  
     aerobus::ContinuedFraction< a0 >, 45  
     aerobus::ContinuedFraction< a0, rest... >, 46  
     aerobus::polynomial< Ring >::val< coeffN  
         >::coeff\_at< index, std::enable\_if\_t<(index<  
         0 || index > 0)>>, 43  
     aerobus::polynomial< Ring >::val< coeffN  
         >::coeff\_at< index, std::enable\_if\_t<(index==0)>  
         >, 44  
     aerobus::Quotient< Ring, X >::val< V >, 80  
     aerobus::type\_list< Ts >::pop\_front, 62  
 type\_at\_t  
     aerobus::internal, 36  
  
 v  
     aerobus::i32::val< x >, 74  
     aerobus::i64::val< x >, 76  
     aerobus::zpz< p >::val< x >, 82  
 vadd\_t  
     aerobus, 30  
 val  
     aerobus::ContinuedFraction< a0 >, 45  
     aerobus::ContinuedFraction< a0, rest... >, 46  
 value  
     aerobus::is\_prime< n >, 55  
 vmul\_t  
     aerobus, 30  
  
 X  
     aerobus::polynomial< Ring >, 61  
  
 zero  
     aerobus::i32, 49  
     aerobus::i64, 53  
     aerobus::polynomial< Ring >, 61  
     aerobus::Quotient< Ring, X >, 66  
     aerobus::zpz< p >, 89