

Aerobus

v1.2

Generated by Doxygen 1.9.8

| | |
|---|-----------|
| 1 Introduction | 1 |
| 1.1 Structures | 1 |
| 1.1.1 Predefined discrete euclidean domains | 1 |
| 1.1.2 Polynomials | 2 |
| 1.1.3 Known polynomials | 2 |
| 1.1.4 Conway polynomials | 3 |
| 1.1.5 Taylor series | 3 |
| 1.2 Operations | 4 |
| 1.2.1 Field of fractions | 4 |
| 1.2.2 Quotient | 5 |
| 1.3 Misc | 5 |
| 1.3.1 Continued Fractions | 5 |
| 2 Namespace Index | 7 |
| 2.1 Namespace List | 7 |
| 3 Concept Index | 9 |
| 3.1 Concepts | 9 |
| 4 Class Index | 11 |
| 4.1 Class List | 11 |
| 5 File Index | 13 |
| 5.1 File List | 13 |
| 6 Namespace Documentation | 15 |
| 6.1 aerobus Namespace Reference | 15 |
| 6.1.1 Detailed Description | 18 |
| 6.1.2 Typedef Documentation | 19 |
| 6.1.2.1 abs_t | 19 |
| 6.1.2.2 addfractions_t | 19 |
| 6.1.2.3 alternate_t | 19 |
| 6.1.2.4 asin | 19 |
| 6.1.2.5 asinh | 20 |
| 6.1.2.6 atan | 20 |
| 6.1.2.7 atanh | 20 |
| 6.1.2.8 bernoulli_t | 20 |
| 6.1.2.9 combination_t | 21 |
| 6.1.2.10 cos | 21 |
| 6.1.2.11 cosh | 21 |
| 6.1.2.12 E_fraction | 21 |
| 6.1.2.13 exp | 22 |
| 6.1.2.14 expm1 | 22 |
| 6.1.2.15 factorial_t | 22 |

| | |
|---|----|
| 6.1.2.16 fpq32 | 22 |
| 6.1.2.17 fpq64 | 23 |
| 6.1.2.18 FractionField | 23 |
| 6.1.2.19 gcd_t | 23 |
| 6.1.2.20 geometric_sum | 23 |
| 6.1.2.21 lnp1 | 23 |
| 6.1.2.22 make_q32_t | 24 |
| 6.1.2.23 make_q64_t | 24 |
| 6.1.2.24 makefraction_t | 24 |
| 6.1.2.25 mulfractions_t | 24 |
| 6.1.2.26 pi64 | 25 |
| 6.1.2.27 PI_fraction | 25 |
| 6.1.2.28 pow_t | 25 |
| 6.1.2.29 pq64 | 25 |
| 6.1.2.30 q32 | 25 |
| 6.1.2.31 q64 | 26 |
| 6.1.2.32 sin | 26 |
| 6.1.2.33 sinh | 26 |
| 6.1.2.34 SQRT2_fraction | 26 |
| 6.1.2.35 SQRT3_fraction | 26 |
| 6.1.2.36 stirling_signed_t | 26 |
| 6.1.2.37 stirling_unsigned_t | 27 |
| 6.1.2.38 tan | 27 |
| 6.1.2.39 tanh | 27 |
| 6.1.2.40 taylor | 28 |
| 6.1.2.41 vadd_t | 28 |
| 6.1.2.42 vmul_t | 28 |
| 6.1.3 Function Documentation | 28 |
| 6.1.3.1 aligned_malloc() | 28 |
| 6.1.3.2 field() | 29 |
| 6.1.4 Variable Documentation | 29 |
| 6.1.4.1 alternate_v | 29 |
| 6.1.4.2 bernoulli_v | 29 |
| 6.1.4.3 combination_v | 30 |
| 6.1.4.4 factorial_v | 30 |
| 6.2 aerobus::internal Namespace Reference | 30 |
| 6.2.1 Detailed Description | 33 |
| 6.2.2 Typedef Documentation | 33 |
| 6.2.2.1 make_index_sequence_reverse | 33 |
| 6.2.2.2 type_at_t | 33 |
| 6.2.3 Function Documentation | 34 |
| 6.2.3.1 index_sequence_reverse() | 34 |

| | |
|--|-----------|
| 6.2.4 Variable Documentation | 34 |
| 6.2.4.1 is_instantiation_of_v | 34 |
| 6.3 aerobus::known_polynomials Namespace Reference | 34 |
| 6.3.1 Detailed Description | 35 |
| 6.3.2 Typedef Documentation | 35 |
| 6.3.2.1 bernoulli | 35 |
| 6.3.2.2 bernstein | 35 |
| 6.3.2.3 chebyshev_T | 35 |
| 6.3.2.4 chebyshev_U | 36 |
| 6.3.2.5 hermite_phys | 36 |
| 6.3.2.6 hermite_prob | 37 |
| 6.3.2.7 laguerre | 37 |
| 6.3.2.8 legendre | 37 |
| 6.3.3 Enumeration Type Documentation | 38 |
| 6.3.3.1 hermite_kind | 38 |
| 7 Concept Documentation | 39 |
| 7.1 aerobus::IsEuclideanDomain Concept Reference | 39 |
| 7.1.1 Concept definition | 39 |
| 7.1.2 Detailed Description | 39 |
| 7.2 aerobus::IsField Concept Reference | 39 |
| 7.2.1 Concept definition | 39 |
| 7.2.2 Detailed Description | 40 |
| 7.3 aerobus::IsRing Concept Reference | 40 |
| 7.3.1 Concept definition | 40 |
| 7.3.2 Detailed Description | 40 |
| 8 Class Documentation | 41 |
| 8.1 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E > Struct Template Reference | 41 |
| 8.2 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0 index > 0)> > Struct Template Reference | 41 |
| 8.2.1 Member Typedef Documentation | 41 |
| 8.2.1.1 type | 41 |
| 8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference | 42 |
| 8.3.1 Member Typedef Documentation | 42 |
| 8.3.1.1 type | 42 |
| 8.4 aerobus::ContinuedFraction< values > Struct Template Reference | 42 |
| 8.5 aerobus::ContinuedFraction< a0 > Struct Template Reference | 42 |
| 8.5.1 Detailed Description | 42 |
| 8.5.2 Member Typedef Documentation | 43 |
| 8.5.2.1 type | 43 |
| 8.5.3 Member Data Documentation | 43 |

| | |
|---|----|
| 8.5.3.1 val | 43 |
| 8.6 aerobus::ContinuedFraction< a0, rest... > Struct Template Reference | 43 |
| 8.6.1 Detailed Description | 43 |
| 8.6.2 Member Typedef Documentation | 44 |
| 8.6.2.1 type | 44 |
| 8.6.3 Member Data Documentation | 44 |
| 8.6.3.1 val | 44 |
| 8.7 aerobus::ConwayPolynomial Struct Reference | 44 |
| 8.8 aerobus::i32 Struct Reference | 44 |
| 8.8.1 Detailed Description | 45 |
| 8.8.2 Member Typedef Documentation | 46 |
| 8.8.2.1 add_t | 46 |
| 8.8.2.2 div_t | 46 |
| 8.8.2.3 eq_t | 46 |
| 8.8.2.4 gcd_t | 46 |
| 8.8.2.5 gt_t | 46 |
| 8.8.2.6 inject_constant_t | 46 |
| 8.8.2.7 inject_ring_t | 46 |
| 8.8.2.8 inner_type | 46 |
| 8.8.2.9 lt_t | 46 |
| 8.8.2.10 mod_t | 46 |
| 8.8.2.11 mul_t | 47 |
| 8.8.2.12 one | 47 |
| 8.8.2.13 pos_t | 47 |
| 8.8.2.14 sub_t | 47 |
| 8.8.2.15 zero | 47 |
| 8.8.3 Member Data Documentation | 47 |
| 8.8.3.1 eq_v | 47 |
| 8.8.3.2 is_euclidean_domain | 47 |
| 8.8.3.3 is_field | 48 |
| 8.8.3.4 pos_v | 48 |
| 8.9 aerobus::i64 Struct Reference | 48 |
| 8.9.1 Detailed Description | 49 |
| 8.9.2 Member Typedef Documentation | 49 |
| 8.9.2.1 add_t | 49 |
| 8.9.2.2 div_t | 49 |
| 8.9.2.3 eq_t | 49 |
| 8.9.2.4 gcd_t | 50 |
| 8.9.2.5 gt_t | 50 |
| 8.9.2.6 inject_constant_t | 50 |
| 8.9.2.7 inject_ring_t | 50 |
| 8.9.2.8 inner_type | 50 |

| | |
|--|----|
| 8.9.2.9 lt_t | 50 |
| 8.9.2.10 mod_t | 50 |
| 8.9.2.11 mul_t | 51 |
| 8.9.2.12 one | 51 |
| 8.9.2.13 pos_t | 51 |
| 8.9.2.14 sub_t | 51 |
| 8.9.2.15 zero | 51 |
| 8.9.3 Member Data Documentation | 51 |
| 8.9.3.1 eq_v | 51 |
| 8.9.3.2 gt_v | 51 |
| 8.9.3.3 is_euclidean_domain | 52 |
| 8.9.3.4 is_field | 52 |
| 8.9.3.5 lt_v | 52 |
| 8.9.3.6 pos_v | 52 |
| 8.10 aerobus::is_prime< n > Struct Template Reference | 52 |
| 8.10.1 Detailed Description | 52 |
| 8.10.2 Member Data Documentation | 53 |
| 8.10.2.1 value | 53 |
| 8.11 aerobus::polynomial< Ring > Struct Template Reference | 53 |
| 8.11.1 Detailed Description | 54 |
| 8.11.2 Member Typedef Documentation | 55 |
| 8.11.2.1 add_t | 55 |
| 8.11.2.2 derive_t | 55 |
| 8.11.2.3 div_t | 55 |
| 8.11.2.4 eq_t | 55 |
| 8.11.2.5 gcd_t | 56 |
| 8.11.2.6 gt_t | 56 |
| 8.11.2.7 inject_constant_t | 56 |
| 8.11.2.8 inject_ring_t | 56 |
| 8.11.2.9 lt_t | 57 |
| 8.11.2.10 mod_t | 57 |
| 8.11.2.11 monomial_t | 57 |
| 8.11.2.12 mul_t | 57 |
| 8.11.2.13 one | 58 |
| 8.11.2.14 pos_t | 58 |
| 8.11.2.15 simplify_t | 58 |
| 8.11.2.16 sub_t | 58 |
| 8.11.2.17 X | 59 |
| 8.11.2.18 zero | 59 |
| 8.11.3 Member Data Documentation | 59 |
| 8.11.3.1 is_euclidean_domain | 59 |
| 8.11.3.2 is_field | 59 |

| | |
|---|----|
| 8.11.3.3 pos_v | 59 |
| 8.12 aerobus::type_list< Ts >::pop_front Struct Reference | 59 |
| 8.12.1 Detailed Description | 60 |
| 8.12.2 Member Typedef Documentation | 60 |
| 8.12.2.1 tail | 60 |
| 8.12.2.2 type | 60 |
| 8.13 aerobus::Quotient< Ring, X > Struct Template Reference | 60 |
| 8.13.1 Detailed Description | 61 |
| 8.13.2 Member Typedef Documentation | 62 |
| 8.13.2.1 add_t | 62 |
| 8.13.2.2 div_t | 62 |
| 8.13.2.3 eq_t | 62 |
| 8.13.2.4 inject_constant_t | 62 |
| 8.13.2.5 inject_ring_t | 63 |
| 8.13.2.6 mod_t | 63 |
| 8.13.2.7 mul_t | 63 |
| 8.13.2.8 one | 63 |
| 8.13.2.9 pos_t | 64 |
| 8.13.2.10 zero | 64 |
| 8.13.3 Member Data Documentation | 64 |
| 8.13.3.1 eq_v | 64 |
| 8.13.3.2 is_euclidean_domain | 64 |
| 8.13.3.3 pos_v | 64 |
| 8.14 aerobus::type_list< Ts >::split< index > Struct Template Reference | 65 |
| 8.14.1 Detailed Description | 65 |
| 8.14.2 Member Typedef Documentation | 65 |
| 8.14.2.1 head | 65 |
| 8.14.2.2 tail | 65 |
| 8.15 aerobus::type_list< Ts > Struct Template Reference | 66 |
| 8.15.1 Detailed Description | 66 |
| 8.15.2 Member Typedef Documentation | 67 |
| 8.15.2.1 at | 67 |
| 8.15.2.2 concat | 67 |
| 8.15.2.3 insert | 67 |
| 8.15.2.4 push_back | 67 |
| 8.15.2.5 push_front | 68 |
| 8.15.2.6 remove | 68 |
| 8.15.3 Member Data Documentation | 68 |
| 8.15.3.1 length | 68 |
| 8.16 aerobus::type_list<> Struct Reference | 68 |
| 8.16.1 Detailed Description | 69 |
| 8.16.2 Member Typedef Documentation | 69 |

| | |
|---|----|
| 8.16.2.1 concat | 69 |
| 8.16.2.2 insert | 69 |
| 8.16.2.3 push_back | 69 |
| 8.16.2.4 push_front | 69 |
| 8.16.3 Member Data Documentation | 70 |
| 8.16.3.1 length | 70 |
| 8.17 aerobus::i32::val< x > Struct Template Reference | 70 |
| 8.17.1 Detailed Description | 70 |
| 8.17.2 Member Typedef Documentation | 71 |
| 8.17.2.1 enclosing_type | 71 |
| 8.17.2.2 is_zero_t | 71 |
| 8.17.3 Member Function Documentation | 71 |
| 8.17.3.1 eval() | 71 |
| 8.17.3.2 get() | 71 |
| 8.17.3.3 to_string() | 72 |
| 8.17.4 Member Data Documentation | 72 |
| 8.17.4.1 v | 72 |
| 8.18 aerobus::i64::val< x > Struct Template Reference | 72 |
| 8.18.1 Detailed Description | 72 |
| 8.18.2 Member Typedef Documentation | 73 |
| 8.18.2.1 enclosing_type | 73 |
| 8.18.2.2 is_zero_t | 73 |
| 8.18.3 Member Function Documentation | 73 |
| 8.18.3.1 eval() | 73 |
| 8.18.3.2 get() | 73 |
| 8.18.3.3 to_string() | 74 |
| 8.18.4 Member Data Documentation | 74 |
| 8.18.4.1 v | 74 |
| 8.19 aerobus::polynomial< Ring >::val< coeffN, coeffs > Struct Template Reference | 74 |
| 8.19.1 Detailed Description | 75 |
| 8.19.2 Member Typedef Documentation | 75 |
| 8.19.2.1 aN | 75 |
| 8.19.2.2 coeff_at_t | 75 |
| 8.19.2.3 enclosing_type | 76 |
| 8.19.2.4 is_zero_t | 76 |
| 8.19.2.5 strip | 76 |
| 8.19.3 Member Function Documentation | 76 |
| 8.19.3.1 eval() | 76 |
| 8.19.3.2 to_string() | 77 |
| 8.19.4 Member Data Documentation | 77 |
| 8.19.4.1 degree | 77 |
| 8.19.4.2 is_zero_v | 77 |

| | |
|---|----|
| 8.20 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference | 77 |
| 8.20.1 Detailed Description | 77 |
| 8.20.2 Member Typedef Documentation | 78 |
| 8.20.2.1 type | 78 |
| 8.21 aerobus::zpz< p >::val< x > Struct Template Reference | 78 |
| 8.21.1 Member Typedef Documentation | 78 |
| 8.21.1.1 enclosing_type | 78 |
| 8.21.1.2 is_zero_t | 79 |
| 8.21.2 Member Function Documentation | 79 |
| 8.21.2.1 eval() | 79 |
| 8.21.2.2 get() | 79 |
| 8.21.2.3 to_string() | 79 |
| 8.21.3 Member Data Documentation | 79 |
| 8.21.3.1 v | 79 |
| 8.22 aerobus::polynomial< Ring >::val< coeffN > Struct Template Reference | 79 |
| 8.22.1 Detailed Description | 80 |
| 8.22.2 Member Typedef Documentation | 80 |
| 8.22.2.1 aN | 80 |
| 8.22.2.2 coeff_at_t | 81 |
| 8.22.2.3 enclosing_type | 81 |
| 8.22.2.4 is_zero_t | 81 |
| 8.22.2.5 strip | 81 |
| 8.22.3 Member Function Documentation | 81 |
| 8.22.3.1 eval() | 81 |
| 8.22.3.2 to_string() | 81 |
| 8.22.4 Member Data Documentation | 82 |
| 8.22.4.1 degree | 82 |
| 8.22.4.2 is_zero_v | 82 |
| 8.23 aerobus::zpz< p > Struct Template Reference | 82 |
| 8.23.1 Detailed Description | 83 |
| 8.23.2 Member Typedef Documentation | 83 |
| 8.23.2.1 add_t | 83 |
| 8.23.2.2 div_t | 84 |
| 8.23.2.3 eq_t | 84 |
| 8.23.2.4 gcd_t | 84 |
| 8.23.2.5 gt_t | 85 |
| 8.23.2.6 inject_constant_t | 85 |
| 8.23.2.7 inner_type | 85 |
| 8.23.2.8 lt_t | 85 |
| 8.23.2.9 mod_t | 85 |
| 8.23.2.10 mul_t | 86 |
| 8.23.2.11 one | 86 |

| | |
|--|------------|
| 8.23.2.12 pos_t | 86 |
| 8.23.2.13 sub_t | 86 |
| 8.23.2.14 zero | 87 |
| 8.23.3 Member Data Documentation | 87 |
| 8.23.3.1 eq_v | 87 |
| 8.23.3.2 gt_v | 87 |
| 8.23.3.3 is_euclidean_domain | 87 |
| 8.23.3.4 is_field | 87 |
| 8.23.3.5 lt_v | 88 |
| 8.23.3.6 pos_v | 88 |
| 9 File Documentation | 89 |
| 9.1 main_page.md File Reference | 89 |
| 9.2 src/aerobus.h File Reference | 89 |
| 9.3 aerobus.h | 89 |
| 10 Examples | 173 |
| 10.1 QuotientRing | 173 |
| 10.2 type_list | 173 |
| 10.3 i32::template | 173 |
| 10.4 i32::add_t | 174 |
| 10.5 i32::sub_t | 174 |
| 10.6 i32::mul_t | 174 |
| 10.7 i32::div_t | 174 |
| 10.8 i32::gt_t | 175 |
| 10.9 i32::eq_t | 175 |
| 10.10 i32::eq_v | 175 |
| 10.11 i32::gcd_t | 175 |
| 10.12 i32::pos_t | 176 |
| 10.13 i32::pos_v | 176 |
| 10.14 i64::template | 176 |
| 10.15 i64::add_t | 176 |
| 10.16 i64::sub_t | 177 |
| 10.17 i64::mul_t | 177 |
| 10.18 i64::div_t | 177 |
| 10.19 i64::mod_t | 177 |
| 10.20 i64::gt_t | 178 |
| 10.21 i64::lt_t | 178 |
| 10.22 i64::lt_v | 178 |
| 10.23 i64::eq_t | 178 |
| 10.24 i64::eq_v | 179 |
| 10.25 i64::gcd_t | 179 |
| 10.26 i64::pos_t | 179 |

| | |
|--|-----|
| 10.27 i64::pos_v | 179 |
| 10.28 polynomial | 180 |
| 10.29 q32::add_t | 180 |
| 10.30 FractionField | 180 |
| 10.31 aerobus::ContinuedFraction | 180 |
| 10.32 PI_fraction::val | 181 |
| 10.33 E_fraction::val | 181 |

| | |
|--------------|------------|
| Index | 183 |
|--------------|------------|

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 Structures

1.1.1 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>`

1.1.2 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.

1.1.3 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`

1.1.4 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>;
```

1.1.5 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
    vfmadd132pd ymm0, ymm14, ymm1
    vfmadd132pd ymm0, ymm13, ymm15
    vfmadd132pd ymm0, ymm12, ymm15
    vfmadd132pd ymm0, ymm11, ymm15
    vfmadd132pd ymm0, ymm10, ymm15
    vfmadd132pd ymm0, ymm9, ymm15
    vfmadd132pd ymm0, ymm8, ymm15
    vfmadd132pd ymm0, ymm7, ymm15
    vfmadd132pd ymm0, ymm6, ymm15
    vfmadd132pd ymm0, ymm5, ymm15
    vfmadd132pd ymm0, ymm4, ymm15
    vfmadd132pd ymm0, ymm3, ymm15
    vfmadd132pd ymm0, ymm2, ymm15
    vmovupd YMMWORD PTR [rdx+rax], ymm0
    add     rax, 32
    cmp     rcx, rax
    jne     .L3
    mov     rax, rdi
    and     rax, -4
    vzeroupper
```

1.2 Operations

1.2.1 Field of fractions

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

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

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

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

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

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

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

Which also have an evaluation function, as polynomial do.

1.2.2 Quotient

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

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

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

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

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

1.3 Misc

1.3.1 Continued Fractions

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

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

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

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


Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

| | | |
|--|--|----|
| aerobus | Main namespace for all publicly exposed types or functions | 15 |
| aerobus::internal | Internal implementations, subject to breaking changes without notice | 30 |
| aerobus::known_polynomials | Families of well known polynomials such as Hermite or Bernstein | 34 |

Chapter 3

Concept Index

3.1 Concepts

Here is a list of all concepts with brief descriptions:

| | |
|---|----|
| aerobus::IsEuclideanDomain | |
| Concept to express R is an euclidean domain | 39 |
| aerobus::IsField | |
| Concept to express R is a field | 39 |
| aerobus::IsRing | |
| Concept to express R is a Ring | 40 |

Chapter 4

Class Index

4.1 Class List

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

| | |
|---|----|
| aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E > | 41 |
| aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0 index > 0)> > | 41 |
| aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > | 42 |
| aerobus::ContinuedFraction< values > | 42 |
| aerobus::ContinuedFraction< a0 > | |
| Specialization for only one coefficient, technically just 'a0' | 42 |
| aerobus::ContinuedFraction< a0, rest... > | |
| Specialization for multiple coefficients (strictly more than one) | 43 |
| aerobus::ConwayPolynomial | 44 |
| aerobus::i32 | |
| 32 bits signed integers, seen as a algebraic ring with related operations | 44 |
| aerobus::i64 | |
| 64 bits signed integers, seen as a algebraic ring with related operations | 48 |
| aerobus::is_prime< n > | |
| Checks if n is prime | 52 |
| aerobus::polynomial< Ring > | 53 |
| aerobus::type_list< Ts >::pop_front | |
| Removes types from head of the list | 59 |
| 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 $\mathbb{Z}/2\mathbb{Z}$ | 60 |
| aerobus::type_list< Ts >::split< index > | |
| Splits list at index | 65 |
| aerobus::type_list< Ts > | |
| Empty pure template struct to handle type list | 66 |
| aerobus::type_list<> | |
| Specialization for empty type list | 68 |
| aerobus::i32::val< x > | |
| Values in i32 , again represented as types | 70 |
| aerobus::i64::val< x > | |
| Values in i64 | 72 |
| aerobus::polynomial< Ring >::val< coeffN, coeffs > | |
| Values (seen as types) in polynomial ring | 74 |
| aerobus::Quotient< Ring, X >::val< V > | |
| Projection values in the quotient ring | 77 |

| | |
|--|----|
| aerobus::zpz< p >::val< x > | 78 |
| aerobus::polynomial< Ring >::val< coeffN > Specialization for constants | 79 |
| aerobus::zpz< p > | 82 |

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

| | |
|--|----|
| src/ aerobus.h | 89 |
|--|----|

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 ($v1 + v2 + \dots + vn$) vals must have same "enclosing_type" and "enclosing_type" must have an `add_t` binary operator
- `template<typename... vals>`
`using vmul_t = typename internal::vmul< vals... >::type`
multiplies multiple values ($v1 + v2 + \dots + vn$) vals must have same "enclosing_type" and "enclosing_type" must have an `mul_t` binary operator
- `template<typename val >`
`using abs_t = std::conditional_t< val::enclosing_type::template pos_v< val >, val, typename val::enclosing_type::template sub_t< typename val::enclosing_type::zero, val > >`
computes absolute value of 'val' val must be a 'value' in a Ring satisfying 'IsEuclideanDomain' concept
- `template<typename Ring >`
`using FractionField = typename internal::FractionFieldImpl< Ring >::type`
- `using q32 = FractionField< i32 >`
32 bits rationals rationals with 32 bits numerator and denominator
- `using fpq32 = FractionField< polynomial< q32 > >`
rational fractions with 32 bits rational coefficients rational fractions with rationals coefficients (32 bits numerator and denominator)
- `using q64 = FractionField< i64 >`
64 bits rationals rationals with 64 bits numerator and denominator
- `using pi64 = polynomial< i64 >`
polynomial with 64 bits integers coefficients
- `using pq64 = polynomial< q64 >`
polynomial with 64 bits rationals coefficients
- `using fpq64 = FractionField< polynomial< q64 > >`
polynomial with 64 bits rational coefficients
- `template<typename Ring , typename v1 , typename v2 >`
`using makefraction_t = typename FractionField< Ring >::template val< v1, v2 >`
helper type : the rational $V1/V2$ in the field of fractions of Ring
- `template<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 , 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 , auto p, auto n>`
`using pow_t = typename internal::pow< T, p, n >::type`
 p^n (as 'val' type in T)
- `template<typename T , template< typename, size_t index > typename coeff_at, size_t deg>`
`using taylor = typename internal::make_taylor_impl< T, coeff_at, internal::make_index_sequence_reverse<`
`deg+1 > >::type`
- `template<typename Integers , size_t deg>`
`using exp = taylor< Integers, internal::exp_coeff, deg >`
 e^x
- `template<typename Integers , size_t deg>`
`using expm1 = typename polynomial< FractionField< Integers > >::template sub_t< exp< Integers, deg`
`>, typename polynomial< FractionField< Integers > >::one >`
 $e^x - 1$
- `template<typename Integers , size_t deg>`
`using ln1 = taylor< Integers, internal::ln1_coeff, deg >`
 $\ln(1 + x)$
- `template<typename Integers , size_t deg>`
`using atan = taylor< Integers, internal::atan_coeff, deg >`
 $\arctan(x)$
- `template<typename Integers , size_t deg>`
`using sin = taylor< Integers, internal::sin_coeff, deg >`
 $\sin(x)$
- `template<typename Integers , size_t deg>`
`using sinh = taylor< Integers, internal::sh_coeff, deg >`
 $\sinh(x)$
- `template<typename Integers , size_t deg>`
`using cosh = taylor< Integers, internal::cosh_coeff, deg >`
 $\cosh(x)$ *hyperbolic cosine*
- `template<typename Integers , size_t deg>`
`using cos = taylor< Integers, internal::cos_coeff, deg >`

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

Functions

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

Variables

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

6.1.1 Detailed Description

main namespace for all publicly exposed types or functions

6.1.2 Typedef Documentation

6.1.2.1 abs_t

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

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

Template Parameters

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

6.1.2.2 addfractions_t

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

helper type : adds two fractions

Template Parameters

| | |
|-------------|--------------------------------|
| <i>Ring</i> | |
| <i>v1</i> | belongs to FractionField<Ring> |
| <i>v2</i> | belongs to FractionField<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 i64) |
| <i>deg</i> | taylor approximation degree |

6.1.2.5 asinh

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

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

Template Parameters

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

6.1.2.6 atan

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

$\operatorname{arctan}(x)$

Template Parameters

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

6.1.2.7 atanh

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

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

Template Parameters

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

6.1.2.8 bernoulli_t

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


nth bernoulli number as type in T

Template Parameters

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

6.1.2.9 combination_t

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

computes binomial coefficient (k among n) as type

Template Parameters

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

6.1.2.10 cos

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

$\cos(x)$ cosinus

Template Parameters

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

6.1.2.11 cosh

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

$\cosh(x)$ hyperbolic cosine

Template Parameters

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

6.1.2.12 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>
```

6.1.2.13 exp

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

$$e^x$$

Template Parameters

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

6.1.2.14 expm1

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

$$e^x - 1$$

Template Parameters

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

6.1.2.15 factorial_t

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

computes factorial(i), as type

Template Parameters

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

6.1.2.16 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)

6.1.2.17 fpq64

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

polynomial with 64 bits rational coefficients

6.1.2.18 FractionField

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

6.1.2.19 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) |
|----------|--------------------------------------|

6.1.2.20 geometric_sum

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

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

Template Parameters

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

6.1.2.21 ln1

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

$\ln(1+x)$

Template Parameters

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

6.1.2.22 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

| | |
|-----|-------------|
| p | numerator |
| q | denominator |

6.1.2.23 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

| | |
|-----|-------------|
| p | numerator |
| q | denominator |

6.1.2.24 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

| | |
|--------|-----------------|
| $Ring$ | the base ring |
| $v1$ | value 1 in Ring |
| $v2$ | value 2 in Ring |

6.1.2.25 mulfractions_t

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

helper type : multiplies two fractions

Template Parameters

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

6.1.2.26 pi64

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

polynomial with 64 bits integers coefficients

6.1.2.27 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>
```

6.1.2.28 pow_t

```
template<typename T , auto p, auto 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> | (from T::inner_type, such as int64_t) |
| <i>n</i> | (from T::inner_type, such as int64_t) |

6.1.2.29 pq64

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

polynomial with 64 bits rationals coefficients

6.1.2.30 q32

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

32 bits rationals rationals with 32 bits numerator and denominator

6.1.2.31 q64

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

64 bits rationals rationals with 64 bits numerator and denominator

6.1.2.32 sin

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

$\sin(x)$

Template Parameters

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

6.1.2.33 sinh

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

$\sinh(x)$

Template Parameters

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

6.1.2.34 SQRT2_fraction

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

approximation of $\sqrt{2}$

6.1.2.35 SQRT3_fraction

```
using aerobus::SQRT3_fraction = typedef ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1>
```

approximation of

6.1.2.36 stirling_signed_t

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

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

Template Parameters

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

6.1.2.37 `stirling_unsigned_t`

```
template<typename T , int n, int k>
using aerobus::stirling\_unsigned\_t = typedef abs\_t<typename internal::stirling_helper<T, n,
k>::type>
```

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

Template Parameters

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

6.1.2.38 `tan`

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

$\tan(x)$ tangent

Template Parameters

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

6.1.2.39 `tanh`

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

$\tanh(x)$ hyperbolic tangent

Template Parameters

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

6.1.2.40 `taylor`

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

Template Parameters

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

6.1.2.41 `vadd_t`

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

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

Template Parameters

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

6.1.2.42 `vmul_t`

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

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

Template Parameters

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

6.1.3 Function Documentation

6.1.3.1 `aligned_malloc()`

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

'portable' aligned allocation of count elements of type T

Template Parameters

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

Parameters

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

6.1.3.2 field()

brief Conway polynomials tparam p characteristic of the aerobus::field (
prime number)

6.1.4 Variable Documentation

6.1.4.1 alternate_v

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

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

Template Parameters

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

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<Float↔
Type> [inline], [constexpr]
```

nth bernoulli number as value in FloatType

Template Parameters

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

6.1.4.3 combination_v

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

computes binomial coefficients (k among n) as value

Template Parameters

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

6.1.4.4 factorial_v

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

computes factorial(i) as value in T

Template Parameters

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

6.2 aerobus::internal Namespace Reference

internal implementations, subject to breaking changes without notice

Classes

- struct **_FractionField**
- struct **_FractionField< Ring, std::enable_if_t< Ring::is_euclidean_domain > >**
- struct **_is_prime**
- struct **_is_prime< 0, i >**
- struct **_is_prime< 1, i >**
- struct **_is_prime< 2, i >**
- struct **_is_prime< 3, i >**
- struct **_is_prime< 5, i >**
- struct **_is_prime< 7, i >**
- struct **_is_prime< n, i, std::enable_if_t<(n !=2 &&n !=3 &&n % 2 !=0 &&n % 3==0)> >**
- struct **_is_prime< n, i, std::enable_if_t<(n !=2 &&n % 2==0)> >**
- struct **_is_prime< n, i, std::enable_if_t<(n % i==0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&i *i > n)> >**
- struct **_is_prime< n, i, std::enable_if_t<(n %(i+2) !=0 &&n % i !=0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&(i *i<=n))> >**

- struct **_is_prime**< n, i, std::enable_if_t<(n % (i+2) == 0 && n >= 9 && n % 3 != 0 && n % 2 != 0 && i * i <= n)>
>
- struct **_is_prime**< n, i, std::enable_if_t<(n >= 9 && i * i > n)> >
- struct **alternate**
- struct **alternate**< T, k, std::enable_if_t<k % 2 != 0> >
- struct **alternate**< T, k, std::enable_if_t<k % 2 == 0> >
- struct **asin_coeff**
- struct **asin_coeff_helper**
- struct **asin_coeff_helper**< T, i, std::enable_if_t<(i & 1) == 0> >
- struct **asin_coeff_helper**< T, i, std::enable_if_t<(i & 1) == 1> >
- struct **asinh_coeff**
- struct **asinh_coeff_helper**
- struct **asinh_coeff_helper**< T, i, std::enable_if_t<(i & 1) == 0> >
- struct **asinh_coeff_helper**< T, i, std::enable_if_t<(i & 1) == 1> >
- struct **atan_coeff**
- struct **atan_coeff_helper**
- struct **atan_coeff_helper**< T, i, std::enable_if_t<(i & 1) == 0> >
- struct **atan_coeff_helper**< T, i, std::enable_if_t<(i & 1) == 1> >
- struct **atanh_coeff**
- struct **atanh_coeff_helper**
- struct **atanh_coeff_helper**< T, i, std::enable_if_t<(i & 1) == 0> >
- struct **atanh_coeff_helper**< T, i, std::enable_if_t<(i & 1) == 1> >
- struct **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 >
- struct **bernstein_helper**< i, m, std::enable_if_t<(m > 0) && (i > 0) && (i < m)> >
- struct **bernstein_helper**< i, m, std::enable_if_t<(m > 0) && (i == 0)> >
- struct **bernstein_helper**< i, m, std::enable_if_t<(m > 0) && (i == m)> >
- struct **chebyshev_helper**
- struct **chebyshev_helper**< 1, 0 >
- struct **chebyshev_helper**< 1, 1 >
- struct **chebyshev_helper**< 2, 0 >
- struct **chebyshev_helper**< 2, 1 >
- 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 >
- struct **hermite_helper**< 0, known_polynomials::hermite_kind::probabilist >
- struct **hermite_helper**< 1, known_polynomials::hermite_kind::physicist >
- struct **hermite_helper**< 1, known_polynomials::hermite_kind::probabilist >
- struct **hermite_helper**< deg, known_polynomials::hermite_kind::physicist >
- struct **hermite_helper**< deg, known_polynomials::hermite_kind::probabilist >
- struct **insert_h**
- struct **is_instantiation_of**
- struct **is_instantiation_of**< TT, TT< Ts... > >
- struct **laguerre_helper**
- struct **laguerre_helper**< 0 >
- struct **laguerre_helper**< 1 >
- struct **legendre_helper**
- struct **legendre_helper**< 0 >
- struct **legendre_helper**< 1 >
- struct **lnp1_coeff**
- struct **lnp1_coeff**< T, 0 >
- struct **make_taylor_impl**
- struct **make_taylor_impl**< T, coeff_at, std::integer_sequence< size_t, ls... > >
- struct **pop_front_h**
- struct **pow**
- struct **pow**< T, n, p, std::enable_if_t< p==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 **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>`
using [chebyshev_T](#) = `typename internal::chebyshev_helper< 1, deg >::type`
Chebyshev polynomials of first kind.
- `template<size_t deg>`
using [chebyshev_U](#) = `typename internal::chebyshev_helper< 2, deg >::type`
Chebyshev polynomials of second kind.
- `template<size_t deg>`
using [laguerre](#) = `typename internal::laguerre_helper< deg >::type`
Laguerre polynomials.
- `template<size_t deg>`
using [hermite_prob](#) = `typename internal::hermite_helper< deg, hermite_kind::probabilist >::type`
Hermite polynomials - probabilist form.
- `template<size_t deg>`
using [hermite_phys](#) = `typename internal::hermite_helper< deg, hermite_kind::physicist >::type`
Hermite polynomials - physicist form.
- `template<size_t i, size_t m>`
using [bernstein](#) = `typename internal::bernstein_helper< i, m >::type`
Bernstein polynomials.
- `template<size_t deg>`
using [legendre](#) = `typename internal::legendre_helper< deg >::type`
Legendre polynomials.
- `template<size_t deg>`
using [bernoulli](#) = `taylor< i64, 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>
using aerobus::known_polynomials::bernoulli = typedef taylor<i64, internal::bernoulli_coeff<deg>>↵
::template inner, deg>
```

Bernoulli polynomials.

See also

[See in Wikipedia](#)

Template Parameters

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

6.3.2.2 bernstein

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

Bernstein polynomials.

See also

[See in Wikipedia](#)

Template Parameters

| | |
|----------|---------------------------------------|
| <i>i</i> | index of polynomial (between 0 and m) |
| <i>m</i> | degree of polynomial |

6.3.2.3 chebyshev_T

```
template<size_t deg>
```

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

Chebyshev polynomials of first kind.

See also

[See in Wikipedia](#)

Template Parameters

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

6.3.2.4 chebyshev_U

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

Chebyshev polynomials of second kind.

See also

[See in Wikipedia](#)

Template Parameters

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

6.3.2.5 hermite_phys

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

Hermite polynomials - physicist form.

See also

[See in Wikipedia](#)

Template Parameters

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

6.3.2.6 hermite_prob

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

Hermite polynomials - probabilist form.

See also

[See in Wikipedia](#)

Template Parameters

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

6.3.2.7 laguerre

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

Laguerre polynomials.

See also

[See in Wikipedia](#)

Template Parameters

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

6.3.2.8 legendre

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

Legendre polynomials.

See also

[See in Wikipedia](#)

Template Parameters

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

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

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 for actual 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 for actual 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 aerobus::i64::val |
| $v2$ | : an element of aerobus::i64::val |

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<2></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 valueType get \(\)](#)
cast x into valueType
- [static std::string to_string \(\)](#)
string representation of value
- [template<typename valueRing >](#)
[static constexpr 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 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 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< x > Struct Template Reference

values in [i64](#)

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

Public Types

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

| | |
|----------|-------------------|
| <i>x</i> | 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 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 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 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 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 coeff_↵
at<index>::type
```

type of coefficient at index

Template Parameters

| | |
|--------------------|--|
| <code>index</code> | |
|--------------------|--|

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 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 valueType get ()`
- `static std::string to_string ()`
- `template<typename valueRing >`
`static constexpr 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 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 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 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

| | |
|---------------------|--|
| <code>coeffN</code> | |
|---------------------|--|

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 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 (booleanvalue)
- `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

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

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

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

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

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

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

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

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 zpz::val |
| v2 | a value in zpz::val |

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 zpz::val |
| v2 | a value in zpz::val |

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

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

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

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

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 zpz::val |
|-----------|-------------------------------------|

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

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

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

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 zpz::val |
| v2 | a value in zpz::val |

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

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

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

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

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

- [src/aerobus.h](#)

Chapter 9

File Documentation

9.1 main_page.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
00016 #ifdef _MSC_VER
00017 #define ALIGNED(x) __declspec(align(x))
00018 #define INLINED __forceinline
00019 #else
00020 #define ALIGNED(x) __attribute__((aligned(x)))
00021 #define INLINED __attribute__((always_inline)) inline
00022 #endif
00023
00024
00025
00026
00027
00028
00029
```

```

00031
00032 // aligned allocation
00033 namespace aerobus {
00040     template<typename T>
00041     T* aligned_malloc(size_t count, size_t alignment) {
00042         #ifdef _MSC_VER
00043             return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
00044         #else
00045             return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
00046         #endif
00047     }
00048 } // namespace aerobus
00049
00050 // concepts
00051 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
00063     template <typename R>
00064     concept IsEuclideanDomain = IsRing<R> && requires {
00065         typename R::template div_t<typename R::one, typename R::one>;
00066         typename R::template mod_t<typename R::one, typename R::one>;
00067         typename R::template gcd_t<typename R::one, typename R::one>;
00068         typename R::template eq_t<typename R::one, typename R::one>;
00069         typename R::template pos_t<typename R::one>;
00070
00071         R::template pos_v<typename R::one> == true;
00072         // typename R::template gt_t<typename R::one, typename R::zero>;
00073         R::is_euclidean_domain == true;
00074     };
00075
00077     template<typename R>
00078     concept IsField = IsEuclideanDomain<R> && requires {
00079         R::is_field == true;
00080     };
00081 } // namespace aerobus
00082
00083 // utilities
00084 namespace aerobus {
00085     namespace internal {
00086         template<template<typename...> typename TT, typename T>
00087         struct is_instantiation_of : std::false_type { };
00088
00089         template<template<typename...> typename TT, typename... Ts>
00090         struct is_instantiation_of<TT, TT<Ts...> : std::true_type { };
00091
00092         template<template<typename...> typename TT, typename T>
00093         inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00094
00095         template <int64_t i, typename T, typename... Ts>
00096         struct type_at {
00097             static_assert(i < sizeof...(Ts) + 1, "index out of range");
00098             using type = typename type_at<i - 1, Ts...>::type;
00099         };
00100
00101         template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00102             using type = T;
00103         };
00104
00105         template <size_t i, typename... Ts>
00106         using type_at_t = typename type_at<i, Ts...>::type;
00107
00108
00109         template<size_t n, size_t i, typename E = void>
00110         struct _is_prime {};
00111
00112         template<size_t i>
00113         struct _is_prime<0, i> {
00114             static constexpr bool value = false;
00115         };
00116
00117         template<size_t i>
00118         struct _is_prime<1, i> {
00119             static constexpr bool value = false;
00120         };
00121
00122         template<size_t i>
00123         struct _is_prime<2, i> {
00124             static constexpr bool value = true;
00125         };
00126

```

```

00127     template<size_t i>
00128     struct _is_prime<3, i> {
00129         static constexpr bool value = true;
00130     };
00131
00132     template<size_t i>
00133     struct _is_prime<5, i> {
00134         static constexpr bool value = true;
00135     };
00136
00137     template<size_t i>
00138     struct _is_prime<7, i> {
00139         static constexpr bool value = true;
00140     };
00141
00142     template<size_t n, size_t i>
00143     struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)>> {
00144         static constexpr bool value = false;
00145     };
00146
00147     template<size_t n, size_t i>
00148     struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)>> {
00149         static constexpr bool value = false;
00150     };
00151
00152     template<size_t n, size_t i>
00153     struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)>> {
00154         static constexpr bool value = true;
00155     };
00156
00157     template<size_t n, size_t i>
00158     struct _is_prime<n, i, std::enable_if_t<(
00159         n % i == 0 &&
00160         n >= 9 &&
00161         n % 3 != 0 &&
00162         n % 2 != 0 &&
00163         i * i > n)>> {
00164         static constexpr bool value = true;
00165     };
00166
00167     template<size_t n, size_t i>
00168     struct _is_prime<n, i, std::enable_if_t<(
00169         n % (i+2) == 0 &&
00170         n >= 9 &&
00171         n % 3 != 0 &&
00172         n % 2 != 0 &&
00173         i * i <= n)>> {
00174         static constexpr bool value = true;
00175     };
00176
00177     template<size_t n, size_t i>
00178     struct _is_prime<n, i, std::enable_if_t<(
00179         n % (i+2) != 0 &&
00180         n % i != 0 &&
00181         n >= 9 &&
00182         n % 3 != 0 &&
00183         n % 2 != 0 &&
00184         (i * i <= n))>> {
00185         static constexpr bool value = _is_prime<n, i+6>::value;
00186     };
00187
00188 } // namespace internal
00189
00190 template<size_t n>
00191 struct is_prime {
00192     static constexpr bool value = internal::_is_prime<n, 5>::value;
00193 };
00194
00195
00196
00197
00201 template<size_t n>
00202 static constexpr bool is_prime_v = is_prime<n>::value;
00203
00204 // gcd
00205 namespace internal {
00206     template <std::size_t... Is>
00207     constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&
00208         -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{}));
00209
00210     template <std::size_t N>
00211     using make_index_sequence_reverse
00212         = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00213
00214     template<typename Ring, typename E = void>
00215     struct gcd;
00216
00217     template<typename Ring>
00218     struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain>> {
00219         template<typename A, typename B, typename E = void>

```

```

00225     struct gcd_helper {};
00226
00227     // B = 0, A > 0
00228     template<typename A, typename B>
00229     struct gcd_helper<A, B, std::enable_if_t<
00230         ((B::is_zero_t::value) &&
00231         (Ring::template gt_t<A, typename Ring::zero>::value))>> {
00232         using type = A;
00233     };
00234
00235     // B = 0, A < 0
00236     template<typename A, typename B>
00237     struct gcd_helper<A, B, std::enable_if_t<
00238         ((B::is_zero_t::value) &&
00239         !(Ring::template gt_t<A, typename Ring::zero>::value))>> {
00240         using type = typename Ring::template sub_t<typename Ring::zero, A>;
00241     };
00242
00243     // B != 0
00244     template<typename A, typename B>
00245     struct gcd_helper<A, B, std::enable_if_t<
00246         (!B::is_zero_t::value)
00247         >> {
00248     private: // NOLINT
00249         // A / B
00250         using k = typename Ring::template div_t<A, B>;
00251         // A - (A/B)*B = A % B
00252         using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B>;
00253
00254     public:
00255         using type = typename gcd_helper<B, m>::type;
00256     };
00257
00258     template<typename A, typename B>
00259     using type = typename gcd_helper<A, B>::type;
00260 };
00261 } // namespace internal
00262
00263 // vadd and vmul
00264 namespace internal {
00265     template<typename... vals>
00266     struct vmul {};
00267
00268     template<typename v1, typename... vals>
00269     struct vmul<v1, vals...> {
00270         using type = typename v1::enclosing_type::template mul_t<v1, typename
vmul<vals...>::type>;
00271     };
00272
00273     template<typename v1>
00274     struct vmul<v1> {
00275         using type = v1;
00276     };
00277
00278     template<typename... vals>
00279     struct vadd {};
00280
00281     template<typename v1, typename... vals>
00282     struct vadd<v1, vals...> {
00283         using type = typename v1::enclosing_type::template add_t<v1, typename
vadd<vals...>::type>;
00284     };
00285
00286     template<typename v1>
00287     struct vadd<v1> {
00288         using type = v1;
00289     };
00290 } // namespace internal
00291
00292 template<typename T, typename A, typename B>
00293 using gcd_t = typename internal::gcd<T>::template type<A, B>;
00294
00295 template<typename... vals>
00296 using vadd_t = typename internal::vadd<vals...>::type;
00297
00298 template<typename... vals>
00299 using vmul_t = typename internal::vmul<vals...>::type;
00300
00301 template<typename val>
00302 requires IsEuclideanDomain<typename val::enclosing_type>
00303 using abs_t = std::conditional_t<
00304     val::enclosing_type::template pos_v<val>,
00305     val, typename val::enclosing_type::template sub_t<typename
val::enclosing_type::zero, val>;
00306 } // namespace aerobus
00307
00308 namespace aerobus {

```



```

00324     template<typename Ring, typename X>
00325     requires IsRing<Ring>
00326     struct Quotient {
00329         template <typename V>
00330         struct val {
00331             public:
00332                 using type = abs_t<typename Ring::template mod_t<V, X>>;
00333         };
00334
00336         using zero = val<typename Ring::zero>;
00337
00339         using one = val<typename Ring::one>;
00340
00344         template<typename v1, typename v2>
00345         using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00346
00350         template<typename v1, typename v2>
00351         using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00352
00356         template<typename v1, typename v2>
00357         using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00358
00362         template<typename v1, typename v2>
00363         using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00364
00368         template<typename v1, typename v2>
00369         using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00370
00374         template<typename v1, typename v2>
00375         static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00376
00380         template<typename v1>
00381         using pos_t = std::true_type;
00382
00386         template<typename v>
00387         static constexpr bool pos_v = pos_t<v>::value;
00388
00390         static constexpr bool is_euclidean_domain = true;
00391
00397         template<auto x>
00398         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00399
00405         template<typename v>
00406         using inject_ring_t = val<v>;
00407     };
00408 } // namespace aerobus
00409
00410 // type_list
00411 namespace aerobus {
00412     template <typename... Ts>
00413     struct type_list;
00414
00416     namespace internal {
00417         template <typename T, typename... Us>
00418         struct pop_front_h {
00419             using tail = type_list<Us...>;
00420             using head = T;
00421         };
00422
00423         template <size_t index, typename L1, typename L2>
00424         struct split_h {
00425             private:
00426                 static_assert(index <= L2::length, "index ouf of bounds");
00427                 using a = typename L2::pop_front::type;
00428                 using b = typename L2::pop_front::tail;
00429                 using c = typename L1::template push_back<a>;
00430
00431             public:
00432                 using head = typename split_h<index - 1, c, b>::head;
00433                 using tail = typename split_h<index - 1, c, b>::tail;
00434         };
00435
00436         template <typename L1, typename L2>
00437         struct split_h<0, L1, L2> {
00438             using head = L1;
00439             using tail = L2;
00440         };
00441
00442         template <size_t index, typename L, typename T>
00443         struct insert_h {
00444             static_assert(index <= L::length, "index ouf of bounds");
00445             using s = typename L::template split<index>;
00446             using left = typename s::head;
00447             using right = typename s::tail;
00448             using ll = typename left::template push_back<T>;
00449             using type = typename ll::template concat<right>;
00450         };

```

```

00451
00452     template <size_t index, typename L>
00453     struct remove_h {
00454         using s = typename L::template split<index>;
00455         using left = typename s::head;
00456         using right = typename s::tail;
00457         using rr = typename right::pop_front::tail;
00458         using type = typename left::template concat<rr>;
00459     };
00460 } // namespace internal
00461
00462 template <typename... Ts>
00463 struct type_list {
00464     private:
00465         template <typename T>
00466         struct concat_h;
00467
00468         template <typename... Us>
00469         struct concat_h<type_list<Us...> {
00470             using type = type_list<Ts..., Us...>;
00471         };
00472
00473     public:
00474         static constexpr size_t length = sizeof...(Ts);
00475
00476         template <typename T>
00477         using push_front = type_list<T, Ts...>;
00478
00479         template <size_t index>
00480         using at = internal::type_at_t<index, Ts...>;
00481
00482         struct pop_front {
00483             using type = typename internal::pop_front_h<Ts...>::head;
00484             using tail = typename internal::pop_front_h<Ts...>::tail;
00485         };
00486
00487         template <typename T>
00488         using push_back = type_list<Ts..., T>;
00489
00490         template <typename U>
00491         using concat = typename concat_h<U>::type;
00492
00493         template <size_t index>
00494         struct split {
00495             private:
00496                 using inner = internal::split_h<index, type_list<>, type_list<Ts...>;
00497
00498             public:
00499                 using head = typename inner::head;
00500                 using tail = typename inner::tail;
00501         };
00502
00503         template <typename T, size_t index>
00504         using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00505
00506         template <size_t index>
00507         using remove = typename internal::remove_h<index, type_list<Ts...>::type;
00508     };
00509
00510 template <>
00511 struct type_list<> {
00512     static constexpr size_t length = 0;
00513
00514     template <typename T>
00515     using push_front = type_list<T>;
00516
00517     template <typename T>
00518     using push_back = type_list<T>;
00519
00520     template <typename U>
00521     using concat = U;
00522
00523     // TODO(jewave): assert index == 0
00524     template <typename T, size_t index>
00525     using insert = type_list<T>;
00526 };
00527 } // namespace aerobus
00528
00529 // i32
00530 namespace aerobus {
00531     struct i32 {
00532         using inner_type = int32_t;
00533         template<int32_t x>
00534         struct val {
00535             using enclosing_type = i32;
00536             static constexpr int32_t v = x;
00537         };
00538     };
00539 }

```

```

00568         template<typename valueType>
00569         static constexpr valueType get() { return static_cast<valueType>(x); }
00570
00572         using is_zero_t = std::bool_constant<x == 0>;
00573
00575         static std::string to_string() {
00576             return std::to_string(x);
00577         }
00578
00581         template<typename valueRing>
00582         static constexpr valueRing eval(const valueRing& v) {
00583             return static_cast<valueRing>(x);
00584         }
00585     };
00586
00588     using zero = val<0>;
00590     using one = val<1>;
00592     static constexpr bool is_field = false;
00594     static constexpr bool is_euclidean_domain = true;
00598     template<auto x>
00599     using inject_constant_t = val<static_cast<int32_t>(x)>;
00600
00601     template<typename v>
00602     using inject_ring_t = v;
00603
00604 private:
00605     template<typename v1, typename v2>
00606     struct add {
00607         using type = val<v1::v + v2::v>;
00608     };
00609
00610     template<typename v1, typename v2>
00611     struct sub {
00612         using type = val<v1::v - v2::v>;
00613     };
00614
00615     template<typename v1, typename v2>
00616     struct mul {
00617         using type = val<v1::v * v2::v>;
00618     };
00619
00620     template<typename v1, typename v2>
00621     struct div {
00622         using type = val<v1::v / v2::v>;
00623     };
00624
00625     template<typename v1, typename v2>
00626     struct remainder {
00627         using type = val<v1::v % v2::v>;
00628     };
00629
00630     template<typename v1, typename v2>
00631     struct gt {
00632         using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00633     };
00634
00635     template<typename v1, typename v2>
00636     struct lt {
00637         using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00638     };
00639
00640     template<typename v1, typename v2>
00641     struct eq {
00642         using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00643     };
00644
00645     template<typename v1>
00646     struct pos {
00647         using type = std::bool_constant<(v1::v > 0)>;
00648     };
00649
00650 public:
00656     template<typename v1, typename v2>
00657     using add_t = typename add<v1, v2>::type;
00658
00664     template<typename v1, typename v2>
00665     using sub_t = typename sub<v1, v2>::type;
00666
00672     template<typename v1, typename v2>
00673     using mul_t = typename mul<v1, v2>::type;
00674
00680     template<typename v1, typename v2>
00681     using div_t = typename div<v1, v2>::type;
00682
00688     template<typename v1, typename v2>
00689     using mod_t = typename remainder<v1, v2>::type;
00690

```

```

00696     template<typename v1, typename v2>
00697     using gt_t = typename gt<v1, v2>::type;
00698
00704     template<typename v1, typename v2>
00705     using lt_t = typename lt<v1, v2>::type;
00706
00712     template<typename v1, typename v2>
00713     using eq_t = typename eq<v1, v2>::type;
00714
00719     template<typename v1, typename v2>
00720     static constexpr bool eq_v = eq_t<v1, v2>::value;
00721
00727     template<typename v1, typename v2>
00728     using gcd_t = gcd_t<i32, v1, v2>;
00729
00734     template<typename v>
00735     using pos_t = typename pos<v>::type;
00736
00741     template<typename v>
00742     static constexpr bool pos_v = pos_t<v>::value;
00743 };
00744 } // namespace aerobus
00745
00746 // i64
00747 namespace aerobus {
00748     struct i64 {
00749         using inner_type = int64_t;
00750         template<int64_t x>
00751         struct val {
00752             using enclosing_type = i64;
00753             static constexpr int64_t v = x;
00754
00755             template<typename valueType>
00756             static constexpr valueType get() {
00757                 return static_cast<valueType>(x);
00758             }
00759
00760             using is_zero_t = std::bool_constant<x == 0>;
00761
00762             static std::string to_string() {
00763                 return std::to_string(x);
00764             }
00765
00766             template<typename valueRing>
00767             static constexpr valueRing eval(const valueRing& v) {
00768                 return static_cast<valueRing>(x);
00769             }
00770         };
00771     };
00772
00773     template<auto x>
00774     using inject_constant_t = val<static_cast<int64_t>(x)>;
00775
00776     template<typename v>
00777     using inject_ring_t = v;
00778
00779     using zero = val<0>;
00780     using one = val<1>;
00781     static constexpr bool is_field = false;
00782     static constexpr bool is_euclidean_domain = true;
00783
00784 private:
00785     template<typename v1, typename v2>
00786     struct add {
00787         using type = val<v1::v + v2::v>;
00788     };
00789
00790     template<typename v1, typename v2>
00791     struct sub {
00792         using type = val<v1::v - v2::v>;
00793     };
00794
00795     template<typename v1, typename v2>
00796     struct mul {
00797         using type = val<v1::v * v2::v>;
00798     };
00799
00800     template<typename v1, typename v2>
00801     struct div {
00802         using type = val<v1::v / v2::v>;
00803     };
00804
00805     template<typename v1, typename v2>
00806     struct remainder {
00807         using type = val<v1::v % v2::v>;
00808     };
00809
00810     template<typename v1, typename v2>
00811 
```

```

00833     struct gt {
00834         using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00835     };
00836
00837     template<typename v1, typename v2>
00838     struct lt {
00839         using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00840     };
00841
00842     template<typename v1, typename v2>
00843     struct eq {
00844         using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00845     };
00846
00847     template<typename v>
00848     struct pos {
00849         using type = std::bool_constant<(v::v > 0)>;
00850     };
00851
00852     public:
00853     template<typename v1, typename v2>
00854     using add_t = typename add<v1, v2>::type;
00855
00856     template<typename v1, typename v2>
00857     using sub_t = typename sub<v1, v2>::type;
00858
00859     template<typename v1, typename v2>
00860     using mul_t = typename mul<v1, v2>::type;
00861
00862     template<typename v1, typename v2>
00863     using div_t = typename div<v1, v2>::type;
00864
00865     template<typename v1, typename v2>
00866     using mod_t = typename remainder<v1, v2>::type;
00867
00868     template<typename v1, typename v2>
00869     using gt_t = typename gt<v1, v2>::type;
00870
00871     template<typename v1, typename v2>
00872     static constexpr bool gt_v = gt_t<v1, v2>::value;
00873
00874     template<typename v1, typename v2>
00875     using lt_t = typename lt<v1, v2>::type;
00876
00877     template<typename v1, typename v2>
00878     static constexpr bool lt_v = lt_t<v1, v2>::value;
00879
00880     template<typename v1, typename v2>
00881     using eq_t = typename eq<v1, v2>::type;
00882
00883     template<typename v1, typename v2>
00884     static constexpr bool eq_v = eq_t<v1, v2>::value;
00885
00886     template<typename v1, typename v2>
00887     using gcd_t = gcd_t<i64, v1, v2>;
00888
00889     template<typename v>
00890     using pos_t = typename pos<v>::type;
00891
00892     template<typename v>
00893     static constexpr bool pos_v = pos_t<v>::value;
00894 };
00895 } // namespace aerobus
00896
00897 // z/pz
00898 namespace aerobus {
00899     template<int32_t p>
00900     struct zp {
00901         using inner_type = int32_t;
00902         template<int32_t x>
00903         struct val {
00904             using enclosing_type = zp<p>;
00905             static constexpr int32_t v = x % p;
00906
00907             template<typename valueType>
00908             static constexpr valueType get() { return static_cast<valueType>(x % p); }
00909
00910             using is_zero_t = std::bool_constant<x % p == 0>;
00911             static std::string to_string() {
00912                 return std::to_string(x % p);
00913             }
00914
00915             template<typename valueRing>
00916             static constexpr valueRing eval(const valueRing& v) {
00917                 return static_cast<valueRing>(x % p);
00918             }
00919         };
00920     };
00921 }

```

```

00989
00990     template<auto x>
00991     using inject_constant_t = val<static_cast<int32_t>(x)>;
00992
00993     using zero = val<0>;
00994     using one = val<1>;
00995     static constexpr bool is_field = is_prime<p>::value;
00996     static constexpr bool is_euclidean_domain = true;
00997
00998 private:
00999     template<typename v1, typename v2>
01000     struct add {
01001         using type = val<(v1::v + v2::v) % p>;
01002     };
01003
01004     template<typename v1, typename v2>
01005     struct sub {
01006         using type = val<(v1::v - v2::v) % p>;
01007     };
01008
01009     template<typename v1, typename v2>
01010     struct mul {
01011         using type = val<(v1::v * v2::v) % p>;
01012     };
01013
01014     template<typename v1, typename v2>
01015     struct div {
01016         using type = val<(v1::v % p) / (v2::v % p)>;
01017     };
01018
01019     template<typename v1, typename v2>
01020     struct remainder {
01021         using type = val<(v1::v % v2::v) % p>;
01022     };
01023
01024     template<typename v1, typename v2>
01025     struct gt {
01026         using type = std::conditional_t<(v1::v % p > v2::v % p), std::true_type, std::false_type>;
01027     };
01028
01029     template<typename v1, typename v2>
01030     struct lt {
01031         using type = std::conditional_t<(v1::v % p < v2::v % p), std::true_type, std::false_type>;
01032     };
01033
01034     template<typename v1, typename v2>
01035     struct eq {
01036         using type = std::conditional_t<(v1::v % p == v2::v % p), std::true_type, std::false_type>;
01037     };
01038
01039     template<typename v1>
01040     struct pos {
01041         using type = std::bool_constant<(v1::v > 0)>;
01042     };
01043
01044 public:
01048     template<typename v1, typename v2>
01049     using add_t = typename add<v1, v2>::type;
01050
01054     template<typename v1, typename v2>
01055     using sub_t = typename sub<v1, v2>::type;
01056
01060     template<typename v1, typename v2>
01061     using mul_t = typename mul<v1, v2>::type;
01062
01066     template<typename v1, typename v2>
01067     using div_t = typename div<v1, v2>::type;
01068
01072     template<typename v1, typename v2>
01073     using mod_t = typename remainder<v1, v2>::type;
01074
01078     template<typename v1, typename v2>
01079     using gt_t = typename gt<v1, v2>::type;
01080
01084     template<typename v1, typename v2>
01085     static constexpr bool gt_v = gt_t<v1, v2>::value;
01086
01090     template<typename v1, typename v2>
01091     using lt_t = typename lt<v1, v2>::type;
01092
01096     template<typename v1, typename v2>
01097     static constexpr bool lt_v = lt_t<v1, v2>::value;
01098
01102     template<typename v1, typename v2>
01103     using eq_t = typename eq<v1, v2>::type;
01104
01108     template<typename v1, typename v2>

```

```

01109     static constexpr bool eq_v = eq_t<v1, v2>::value;
01110
01114     template<typename v1, typename v2>
01115     using gcd_t = gcd_t<i32, v1, v2>;
01116
01119     template<typename v1>
01120     using pos_t = typename pos<v1>::type;
01121
01124     template<typename v>
01125     static constexpr bool pos_v = pos_t<v>::value;
01126 };
01127 } // namespace aerobus
01128
01129 // polynomial
01130 namespace aerobus {
01131     // coeffN x^N + ...
01136     template<typename Ring>
01137     requires IsEuclideanDomain<Ring>
01138     struct polynomial {
01139         static constexpr bool is_field = false;
01140         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01141
01145         template<typename coeffN, typename... coeffs>
01146         struct val {
01148             using enclosing_type = polynomial<Ring>;
01150             static constexpr size_t degree = sizeof...(coeffs);
01152             using aN = coeffN;
01154             using strip = val<coeffs...>;
01156             using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01158             static constexpr bool is_zero_v = is_zero_t::value;
01159
01160         private:
01161             template<size_t index, typename E = void>
01162             struct coeff_at {};
01163
01164             template<size_t index>
01165             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))> {
01166                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01167             };
01168
01169             template<size_t index>
01170             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))> {
01171                 using type = typename Ring::zero;
01172             };
01173
01174         public:
01177             template<size_t index>
01178             using coeff_at_t = typename coeff_at<index>::type;
01179
01182             static std::string to_string() {
01183                 return string_helper<coeffN, coeffs...>::func();
01184             }
01185
01190             template<typename valueRing>
01191             static constexpr valueRing eval(const valueRing& x) {
01192                 return horner_evaluation<valueRing, val>
01193                     ::template inner<0, degree + 1>
01194                     ::func(static_cast<valueRing>(0), x);
01195             }
01196 };
01197
01200     template<typename coeffN>
01201     struct val<coeffN> {
01203         using enclosing_type = polynomial<Ring>;
01205         static constexpr size_t degree = 0;
01206         using aN = coeffN;
01207         using strip = val<coeffN>;
01208         using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01209
01210         static constexpr bool is_zero_v = is_zero_t::value;
01211
01212         template<size_t index, typename E = void>
01213         struct coeff_at {};
01214
01215         template<size_t index>
01216         struct coeff_at<index, std::enable_if_t<(index == 0)> {
01217             using type = aN;
01218         };
01219
01220         template<size_t index>
01221         struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)> {
01222             using type = typename Ring::zero;
01223         };
01224
01225         template<size_t index>
01226         using coeff_at_t = typename coeff_at<index>::type;
01227

```

```

01228         static std::string to_string() {
01229             return string_helper<coeffN>::func();
01230         }
01231
01232         template<typename valueRing>
01233         static constexpr valueRing eval(const valueRing& x) {
01234             return static_cast<valueRing>(aN::template get<valueRing>());
01235         }
01236     };
01237
01239     using zero = val<typename Ring::zero>;
01241     using one = val<typename Ring::one>;
01243     using X = val<typename Ring::one, typename Ring::zero>;
01244
01245 private:
01246     template<typename P, typename E = void>
01247     struct simplify;
01248
01249     template<typename P1, typename P2, typename I>
01250     struct add_low;
01251
01252     template<typename P1, typename P2>
01253     struct add {
01254         using type = typename simplify<typename add_low<
01255             P1,
01256             P2,
01257             internal::make_index_sequence_reverse<
01258                 std::max(P1::degree, P2::degree) + 1
01259             >::type>::type;
01260     };
01261
01262     template<typename P1, typename P2, typename I>
01263     struct sub_low;
01264
01265     template<typename P1, typename P2, typename I>
01266     struct mul_low;
01267
01268     template<typename v1, typename v2>
01269     struct mul {
01270         using type = typename mul_low<
01271             v1,
01272             v2,
01273             internal::make_index_sequence_reverse<
01274                 v1::degree + v2::degree + 1
01275             >::type;
01276     };
01277
01278     template<typename coeff, size_t deg>
01279     struct monomial;
01280
01281     template<typename v, typename E = void>
01282     struct derive_helper {};
01283
01284     template<typename v>
01285     struct derive_helper<v, std::enable_if_t<v::degree == 0> {
01286         using type = zero;
01287     };
01288
01289     template<typename v>
01290     struct derive_helper<v, std::enable_if_t<v::degree != 0> {
01291         using type = typename add<
01292             typename derive_helper<typename simplify<typename v::strip>::type>::type,
01293             typename monomial<
01294                 typename Ring::template mul_t<
01295                     typename v::aN,
01296                     typename Ring::template inject_constant_t<(v::degree)>
01297                 >,
01298                 v::degree - 1
01299             >::type
01300         >::type;
01301     };
01302
01303     template<typename v1, typename v2, typename E = void>
01304     struct eq_helper {};
01305
01306     template<typename v1, typename v2>
01307     struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree> {
01308         using type = std::false_type;
01309     };
01310
01311     template<typename v1, typename v2>
01312     struct eq_helper<v1, v2, std::enable_if_t<
01313         v1::degree == v2::degree &&
01314         (v1::degree != 0 || v2::degree != 0) &&
01315         std::is_same<
01316             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,

```



```

01318         std::false_type
01319         >::value
01320     >
01321     > {
01322         using type = std::false_type;
01323     };
01324
01325     template<typename v1, typename v2>
01326     struct eq_helper<v1, v2, std::enable_if_t<
01327         v1::degree == v2::degree &&
01328         (v1::degree != 0 || v2::degree != 0) &&
01329         std::is_same<
01330             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01331             std::true_type
01332         >::value
01333     > {
01334         using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01335     };
01336
01337     template<typename v1, typename v2>
01338     struct eq_helper<v1, v2, std::enable_if_t<
01339         v1::degree == v2::degree &&
01340         (v1::degree == 0)
01341     > {
01342         using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01343     };
01344
01345     template<typename v1, typename v2, typename E = void>
01346     struct lt_helper {};
01347
01348     template<typename v1, typename v2>
01349     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)>> {
01350         using type = std::true_type;
01351     };
01352
01353     template<typename v1, typename v2>
01354     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)>> {
01355         using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01356     };
01357
01358     template<typename v1, typename v2>
01359     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)>> {
01360         using type = std::false_type;
01361     };
01362
01363     template<typename v1, typename v2, typename E = void>
01364     struct gt_helper {};
01365
01366     template<typename v1, typename v2>
01367     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)>> {
01368         using type = std::true_type;
01369     };
01370
01371     template<typename v1, typename v2>
01372     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)>> {
01373         using type = std::false_type;
01374     };
01375
01376     template<typename v1, typename v2>
01377     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)>> {
01378         using type = std::false_type;
01379     };
01380
01381     // when high power is zero : strip
01382     template<typename P>
01383     struct simplify<P, std::enable_if_t<
01384         std::is_same<
01385             typename Ring::zero,
01386             typename P::aN
01387         >::value && (P::degree > 0)
01388     > {
01389         using type = typename simplify<typename P::strip>::type;
01390     };
01391
01392     // otherwise : do nothing
01393     template<typename P>
01394     struct simplify<P, std::enable_if_t<
01395         !std::is_same<
01396             typename Ring::zero,
01397             typename P::aN
01398         >::value && (P::degree > 0)
01399     > {
01400         using type = P;
01401     };
01402
01403     // do not simplify constants
01404     template<typename P>

```

```

01405     struct simplify<P, std::enable_if_t<P::degree == 0> {
01406         using type = P;
01407     };
01408
01409     // addition at
01410     template<typename P1, typename P2, size_t index>
01411     struct add_at {
01412         using type =
01413             typename Ring::template add_t<
01414                 typename P1::template coeff_at_t<index>,
01415                 typename P2::template coeff_at_t<index>>;
01416     };
01417
01418     template<typename P1, typename P2, size_t index>
01419     using add_at_t = typename add_at<P1, P2, index>::type;
01420
01421     template<typename P1, typename P2, std::size_t... I>
01422     struct add_low<P1, P2, std::index_sequence<I...> {
01423         using type = val<add_at_t<P1, P2, I>...>;
01424     };
01425
01426     // subtraction at
01427     template<typename P1, typename P2, size_t index>
01428     struct sub_at {
01429         using type =
01430             typename Ring::template sub_t<
01431                 typename P1::template coeff_at_t<index>,
01432                 typename P2::template coeff_at_t<index>>;
01433     };
01434
01435     template<typename P1, typename P2, size_t index>
01436     using sub_at_t = typename sub_at<P1, P2, index>::type;
01437
01438     template<typename P1, typename P2, std::size_t... I>
01439     struct sub_low<P1, P2, std::index_sequence<I...> {
01440         using type = val<sub_at_t<P1, P2, I>...>;
01441     };
01442
01443     template<typename P1, typename P2>
01444     struct sub {
01445         using type = typename simplify<typename sub_low<
01446             P1,
01447             P2,
01448             internal::make_index_sequence_reverse<
01449                 std::max(P1::degree, P2::degree) + 1
01450             >::type>::type;
01451     };
01452
01453     // multiplication at
01454     template<typename v1, typename v2, size_t k, size_t index, size_t stop>
01455     struct mul_at_loop_helper {
01456         using type = typename Ring::template add_t<
01457             typename Ring::template mul_t<
01458                 typename v1::template coeff_at_t<index>,
01459                 typename v2::template coeff_at_t<k - index>
01460             >,
01461             typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
01462         >;
01463     };
01464
01465     template<typename v1, typename v2, size_t k, size_t stop>
01466     struct mul_at_loop_helper<v1, v2, k, stop, stop> {
01467         using type = typename Ring::template mul_t<
01468             typename v1::template coeff_at_t<stop>,
01469             typename v2::template coeff_at_t<0>>;
01470     };
01471
01472     template <typename v1, typename v2, size_t k, typename E = void>
01473     struct mul_at {};
01474
01475     template<typename v1, typename v2, size_t k>
01476     struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)> {
01477         using type = typename Ring::zero;
01478     };
01479
01480     template<typename v1, typename v2, size_t k>
01481     struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)> {
01482         using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01483     };
01484
01485     template<typename P1, typename P2, size_t index>
01486     using mul_at_t = typename mul_at<P1, P2, index>::type;
01487
01488     template<typename P1, typename P2, std::size_t... I>
01489     struct mul_low<P1, P2, std::index_sequence<I...> {
01490         using type = val<mul_at_t<P1, P2, I>...>;
01491     };

```

```

01492
01493 // division helper
01494 template< typename A, typename B, typename Q, typename R, typename E = void>
01495 struct div_helper {};
01496
01497 template<typename A, typename B, typename Q, typename R>
01498 struct div_helper<A, B, Q, R, std::enable_if_t<
01499     (R::degree < B::degree) ||
01500     (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)> {
01501     using q_type = Q;
01502     using mod_type = R;
01503     using gcd_type = B;
01504 };
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     private: // NOLINT
01511         using rN = typename R::aN;
01512         using bN = typename B::aN;
01513         using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
01514             B::degree>::type;
01514         using rr = typename sub<R, typename mul<pT, B>::type>::type;
01515         using qq = typename add<Q, pT>::type;
01516
01517     public:
01518         using q_type = typename div_helper<A, B, qq, rr>::q_type;
01519         using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
01520         using gcd_type = rr;
01521 };
01522
01523 template<typename A, typename B>
01524 struct div {
01525     static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
01526     using q_type = typename div_helper<A, B, zero, A>::q_type;
01527     using m_type = typename div_helper<A, B, zero, A>::mod_type;
01528 };
01529
01530 template<typename P>
01531 struct make_unit {
01532     using type = typename div<P, val<typename P::aN>::q_type>;
01533 };
01534
01535 template<typename coeff, size_t deg>
01536 struct monomial {
01537     using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
01538 };
01539
01540 template<typename coeff>
01541 struct monomial<coeff, 0> {
01542     using type = val<coeff>;
01543 };
01544
01545 template<typename valueRing, typename P>
01546 struct horner_evaluation {
01547     template<size_t index, size_t stop>
01548     struct inner {
01549         static constexpr valueRing func(const valueRing& accum, const valueRing& x) {
01550             constexpr valueRing coeff =
01551                 static_cast<valueRing>(P::template coeff_at_t<P::degree - index>::template
01552                     get<valueRing>());
01552             return horner_evaluation<valueRing, P>::template inner<index + 1, stop>::func(x *
01553                 accum + coeff, x);
01554         };
01555     };
01556     template<size_t stop>
01557     struct inner<stop, stop> {
01558         static constexpr valueRing func(const valueRing& accum, const valueRing& x) {
01559             return accum;
01560         };
01561     };
01562 };
01563
01564 template<typename coeff, typename... coeffs>
01565 struct string_helper {
01566     static std::string func() {
01567         std::string tail = string_helper<coeffs...>::func();
01568         std::string result = "";
01569         if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
01570             return tail;
01571         } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
01572             if (sizeof...(coeffs) == 1) {
01573                 result += "x";
01574             } else {
01575                 result += "x^" + std::to_string(sizeof...(coeffs));

```

```

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

```

```

01707     requires IsEuclideanDomain<Ring>
01708     struct _FractionField {};
01709
01710     template<typename Ring>
01711     requires IsEuclideanDomain<Ring>
01712     struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain> {
01713         static constexpr bool is_field = true;
01714         static constexpr bool is_euclidean_domain = true;
01715
01716     private:
01717         template<typename val1, typename val2, typename E = void>
01718         struct to_string_helper {};
01719
01720         template<typename val1, typename val2>
01721         struct to_string_helper <val1, val2,
01722             std::enable_if_t<
01723                 Ring::template eq_t<
01724                     val2, typename Ring::one
01725                 >::value
01726             >
01727         > {
01728             static std::string func() {
01729                 return val1::to_string();
01730             }
01731         };
01732
01733         template<typename val1, typename val2>
01734         struct to_string_helper<val1, val2,
01735             std::enable_if_t<
01736                 !Ring::template eq_t<
01737                     val2,
01738                     typename Ring::one
01739                 >::value
01740             >
01741         > {
01742             static std::string func() {
01743                 return "(" + val1::to_string() + " ) / ( " + val2::to_string() + " )";
01744             }
01745         };
01746
01747     public:
01748         template<typename val1, typename val2>
01749         struct val {
01750             using x = val1;
01751             using y = val2;
01752             using is_zero_t = typename val1::is_zero_t;
01753             static constexpr bool is_zero_v = val1::is_zero_t::value;
01754
01755             using ring_type = Ring;
01756             using enclosing_type = _FractionField<Ring>;
01757
01758             static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
01759
01760             template<typename valueType>
01761             static constexpr valueType get() { return static_cast<valueType>(x::v) /
01762                 static_cast<valueType>(y::v); }
01763
01764             static std::string to_string() {
01765                 return to_string_helper<val1, val2>::func();
01766             }
01767
01768             template<typename valueRing>
01769             static constexpr valueRing eval(const valueRing& v) {
01770                 return x::eval(v) / y::eval(v);
01771             }
01772         };
01773
01774         using zero = val<typename Ring::zero, typename Ring::one>;
01775         using one = val<typename Ring::one, typename Ring::one>;
01776
01777         template<typename v>
01778         using inject_t = val<v, typename Ring::one>;
01779
01780         template<auto x>
01781         using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
01782             Ring::one>;
01783
01784         template<typename v>
01785         using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
01786
01787         using ring_type = Ring;
01788
01789     private:
01790         template<typename v, typename E = void>
01791         struct simplify {};
01792
01793         // x = 0

```

```

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

```

```

01907         public:
01908             using type = typename _FractionField<Ring>::template simplify_t<val<a, b>>;
01909     };
01910
01911     template<typename v1, typename v2>
01912     struct div<v1, v2, std::enable_if_t<
01913         std::is_same<zero, v1>::value && std::is_same<v2, zero>::value> {
01914         using type = one;
01915     };
01916
01917     template<typename v1, typename v2>
01918     struct eq {
01919         using type = std::conditional_t<
01920             std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
01921             std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
01922             std::true_type,
01923             std::false_type>;
01924     };
01925
01926     template<typename v1, typename v2, typename E = void>
01927     struct gt;
01928
01929     template<typename v1, typename v2>
01930     struct gt<v1, v2, std::enable_if_t<
01931         (eq<v1, v2>::type::value)
01932         > {
01933         using type = std::false_type;
01934     };
01935
01936     template<typename v1, typename v2>
01937     struct gt<v1, v2, std::enable_if_t<
01938         (!eq<v1, v2>::type::value) &&
01939         (!pos<v1>::type::value) && (!pos<v2>::type::value)
01940         > {
01941         using type = typename gt<
01942             typename sub<zero, v1>::type, typename sub<zero, v2>::type
01943             >::type;
01944     };
01945
01946     template<typename v1, typename v2>
01947     struct gt<v1, v2, std::enable_if_t<
01948         (!eq<v1, v2>::type::value) &&
01949         (pos<v1>::type::value) && (!pos<v2>::type::value)
01950         > {
01951         using type = std::true_type;
01952     };
01953
01954     template<typename v1, typename v2>
01955     struct gt<v1, v2, std::enable_if_t<
01956         (!eq<v1, v2>::type::value) &&
01957         (!pos<v1>::type::value) && (pos<v2>::type::value)
01958         > {
01959         using type = std::false_type;
01960     };
01961
01962     template<typename v1, typename v2>
01963     struct gt<v1, v2, std::enable_if_t<
01964         (!eq<v1, v2>::type::value) &&
01965         (pos<v1>::type::value) && (pos<v2>::type::value)
01966         > {
01967         using type = typename Ring::template gt_t<
01968             typename Ring::template mul_t<v1::x, v2::y>,
01969             typename Ring::template mul_t<v2::y, v2::x>
01970             >;
01971     };
01972
01973     public:
01974         template<typename v1, typename v2>
01975         using add_t = typename add<v1, v2>::type;
01976
01977         template<typename v1, typename v2>
01978         using mod_t = zero;
01979
01980         template<typename v1, typename v2>
01981         using gcd_t = v1;
01982
01983         template<typename v1, typename v2>
01984         using sub_t = typename sub<v1, v2>::type;
01985
01986         template<typename v1, typename v2>
01987         using mul_t = typename mul<v1, v2>::type;
01988
01989         template<typename v1, typename v2>
01990         using div_t = typename div<v1, v2>::type;
01991
01992         template<typename v1, typename v2>
01993         using eq_t = typename eq<v1, v2>::type;

```

```

02018
02022     template<typename v1, typename v2>
02023     static constexpr bool eq_v = eq<v1, v2>::type::value;
02024
02028     template<typename v1, typename v2>
02029     using gt_t = typename gt<v1, v2>::type;
02030
02034     template<typename v1, typename v2>
02035     static constexpr bool gt_v = gt<v1, v2>::type::value;
02036
02039     template<typename v1>
02040     using pos_t = typename pos<v1>::type;
02041
02044     template<typename v>
02045     static constexpr bool pos_v = pos_t<v>::value;
02046 };
02047
02048     template<typename Ring, typename E = void>
02049     requires IsEuclideanDomain<Ring>
02050     struct FractionFieldImpl {};
02051
02052     // fraction field of a field is the field itself
02053     template<typename Field>
02054     requires IsEuclideanDomain<Field>
02055     struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field> {
02056         using type = Field;
02057         template<typename v>
02058         using inject_t = v;
02059     };
02060
02061     // fraction field of a ring is the actual fraction field
02062     template<typename Ring>
02063     requires IsEuclideanDomain<Ring>
02064     struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field> {
02065         using type = _FractionField<Ring>;
02066     };
02067 } // namespace internal
02068
02072     template<typename Ring>
02073     requires IsEuclideanDomain<Ring>
02074     using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02075 } // namespace aerobus
02076
02077 // short names for common types
02078 namespace aerobus {
02081     using q32 = FractionField<i32>;
02084     using fpq32 = FractionField<polynomial<q32>>;
02087     using q64 = FractionField<i64>;
02089     using pi64 = polynomial<i64>;
02091     using pq64 = polynomial<q64>;
02093     using fpq64 = FractionField<polynomial<q64>>;
02098     template<typename Ring, typename v1, typename v2>
02099     using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02100
02104     template<int64_t p, int64_t q>
02105     using make_q64_t = typename q64::template simplify_t<
02106         typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>>;
02107
02111     template<int32_t p, int32_t q>
02112     using make_q32_t = typename q32::template simplify_t<
02113         typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>>;
02114
02119     template<typename Ring, typename v1, typename v2>
02120     using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02125     template<typename Ring, typename v1, typename v2>
02126     using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02127 } // namespace aerobus
02128
02129 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02130 namespace aerobus {
02131     namespace internal {
02132         template<typename T, size_t x, typename E = void>
02133         struct factorial {};
02134
02135         template<typename T, size_t x>
02136         struct factorial<T, x, std::enable_if_t<(x > 0)> {
02137             private:
02138                 template<typename, size_t, typename>
02139                 friend struct factorial;
02140             public:
02141                 using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
02142 x - 1>::type>;
02143                 static constexpr typename T::inner_type value = type::template get<typename
02144 T::inner_type>();
02145         };
02146     }
02147     template<typename T>

```



```

02146     struct factorial<T, 0> {
02147     public:
02148         using type = typename T::one;
02149         static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02150     };
02151     } // namespace internal
02152
02153     template<typename T, size_t i>
02154     using factorial_t = typename internal::factorial<T, i>::type;
02155
02156     template<typename T, size_t i>
02157     inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02158
02159     namespace internal {
02160         template<typename T, size_t k, size_t n, typename E = void>
02161         struct combination_helper {};
02162
02163         template<typename T, size_t k, size_t n>
02164         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)>> {
02165             using type = typename FractionField<T>::template mul_t<
typename combination_helper<T, k - 1, n - 1>::type,
makefraction_t<T, typename T::template val<n>, typename T::template val<k>>>;
02166         };
02167
02168         template<typename T, size_t k, size_t n>
02169         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)>> {
02170             using type = typename combination_helper<T, n - k, n>::type;
02171         };
02172
02173         template<typename T, size_t n>
02174         struct combination_helper<T, 0, n> {
02175             using type = typename FractionField<T>::one;
02176         };
02177
02178         template<typename T, size_t k, size_t n>
02179         struct combination {
02180             using type = typename internal::combination_helper<T, k, n>::type::x;
02181             static constexpr typename T::inner_type value =
internal::combination_helper<T, k, n>::type::template get<typename
T::inner_type>();
02182         };
02183     } // namespace internal
02184
02185     template<typename T, size_t k, size_t n>
02186     using combination_t = typename internal::combination<T, k, n>::type;
02187
02188     template<typename T, size_t k, size_t n>
02189     inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02190
02191     namespace internal {
02192         template<typename T, size_t m>
02193         struct bernoulli;
02194
02195         template<typename T, typename accum, size_t k, size_t m>
02196         struct bernoulli_helper {
02197             using type = typename bernoulli_helper<
T,
addfractions_t<T,
accum,
mulfractions_t<T,
makefraction_t<T,
combination_t<T, k, m + 1>,
typename T::one>,
typename bernoulli<T, k>::type
>
>,
k + 1,
m>::type;
02198         };
02199
02200         template<typename T, typename accum, size_t m>
02201         struct bernoulli_helper<T, accum, m, m> {
02202             using type = accum;
02203         };
02204
02205         template<typename T, size_t m>
02206         struct bernoulli {
02207             using type = typename FractionField<T>::template mul_t<
typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
makefraction_t<T,
typename T::template val<static_cast<typename T::inner_type>(-1)>,
typename T::template val<static_cast<typename T::inner_type>(m + 1)>
>
>;
02208     };

```

```

02243
02244     template<typename floatType>
02245     static constexpr floatType value = type::template get<floatType>();
02246 };
02247
02248     template<typename T>
02249     struct bernoulli<T, 0> {
02250         using type = typename FractionField<T>::one;
02251
02252         template<typename floatType>
02253         static constexpr floatType value = type::template get<floatType>();
02254     };
02255 } // namespace internal
02256
02260 template<typename T, size_t n>
02261 using bernoulli_t = typename internal::bernoulli<T, n>::type;
02262
02267 template<typename FloatType, typename T, size_t n>
02268 inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02269
02270 namespace internal {
02271     template<typename T, int k, typename E = void>
02272     struct alternate {};
02273
02274     template<typename T, int k>
02275     struct alternate<T, k, std::enable_if_t<k % 2 == 0> {
02276         using type = typename T::one;
02277         static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02278     };
02279
02280     template<typename T, int k>
02281     struct alternate<T, k, std::enable_if_t<k % 2 != 0> {
02282         using type = typename T::template sub_t<typename T::zero, typename T::one>;
02283         static constexpr typename T::inner_type value = type::template get<typename
T::inner_type>();
02284     };
02285 } // namespace internal
02286
02289 template<typename T, int k>
02290 using alternate_t = typename internal::alternate<T, k>::type;
02291
02292 namespace internal {
02293     template<typename T, int n, int k, typename E = void>
02294     struct stirling_helper {};
02295
02296     template<typename T>
02297     struct stirling_helper<T, 0, 0> {
02298         using type = typename T::one;
02299     };
02300
02301     template<typename T, int n>
02302     struct stirling_helper<T, n, 0, std::enable_if_t<(n > 0)> {
02303         using type = typename T::zero;
02304     };
02305
02306     template<typename T, int n>
02307     struct stirling_helper<T, 0, n, std::enable_if_t<(n > 0)> {
02308         using type = typename T::zero;
02309     };
02310
02311     template<typename T, int n, int k>
02312     struct stirling_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)> {
02313         using type = typename T::template sub_t<
02314             typename stirling_helper<T, n-1, k-1>::type,
02315             typename T::template mul_t<
02316                 typename T::template inject_constant_t<n-1>,
02317                 typename stirling_helper<T, n-1, k>::type
02318             >;
02319     };
02320 } // namespace internal
02321
02326 template<typename T, int n, int k>
02327 using stirling_signed_t = typename internal::stirling_helper<T, n, k>::type;
02328
02333 template<typename T, int n, int k>
02334 using stirling_unsigned_t = abs_t<typename internal::stirling_helper<T, n, k>::type>;
02335
02340 template<typename T, int n, int k>
02341 static constexpr typename T::inner_type stirling_signed_v = stirling_signed_t<T, n, k>::v;
02342
02343
02348 template<typename T, int n, int k>
02349 static constexpr typename T::inner_type stirling_unsigned_v = stirling_unsigned_t<T, n, k>::v;
02350
02353 template<typename T, size_t k>
02354 inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;

```

```

02355
02356 namespace internal {
02357     template<typename T, auto p, auto n, typename E = void>
02358     struct pow {};
02359
02360     template<typename T, auto p, auto n>
02361     struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)> {
02362         using type = typename T::template mul_t<
02363             typename pow<T, p, n/2>::type,
02364             typename pow<T, p, n/2>::type
02365         >;
02366     };
02367
02368     template<typename T, auto p, auto n>
02369     struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)> {
02370         using type = typename T::template mul_t<
02371             typename T::template inject_constant_t<p>,
02372             typename T::template mul_t<
02373                 typename pow<T, p, n/2>::type,
02374                 typename pow<T, p, n/2>::type
02375             >
02376         >;
02377     };
02378
02379     template<typename T, auto n, auto p>
02380     struct pow<T, n, p, std::enable_if_t<p == 0> { using type = typename T::one; };
02381 } // namespace internal
02382
02387 template<typename T, auto p, auto n>
02388 using pow_t = typename internal::pow<T, p, n>::type;
02389
02394 template<typename T, auto p, auto n>
02395 static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
02396
02397 namespace internal {
02398     template<typename, template<typename, size_t> typename, class>
02399     struct make_taylor_impl;
02400
02401     template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
02402     struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...> {
02403         using type = typename polynomial<FractionField<T>>::template val<typename coeff_at<T,
02404             Is>::type...>;
02405     };
02406
02411 template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
02412 using taylor = typename internal::make_taylor_impl<
02413     T,
02414     coeff_at,
02415     internal::make_index_sequence_reverse<deg + 1>::type;
02416
02417 namespace internal {
02418     template<typename T, size_t i>
02419     struct exp_coeff {
02420         using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02421     };
02422
02423     template<typename T, size_t i, typename E = void>
02424     struct sin_coeff_helper {};
02425
02426     template<typename T, size_t i>
02427     struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02428         using type = typename FractionField<T>::zero;
02429     };
02430
02431     template<typename T, size_t i>
02432     struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02433         using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02434     };
02435
02436     template<typename T, size_t i>
02437     struct sin_coeff {
02438         using type = typename sin_coeff_helper<T, i>::type;
02439     };
02440
02441     template<typename T, size_t i, typename E = void>
02442     struct sh_coeff_helper {};
02443
02444     template<typename T, size_t i>
02445     struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02446         using type = typename FractionField<T>::zero;
02447     };
02448
02449     template<typename T, size_t i>
02450     struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02451         using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02452     };

```

```

02453
02454     template<typename T, size_t i>
02455     struct sh_coeff {
02456         using type = typename sh_coeff_helper<T, i>::type;
02457     };
02458
02459     template<typename T, size_t i, typename E = void>
02460     struct cos_coeff_helper {};
02461
02462     template<typename T, size_t i>
02463     struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02464         using type = typename FractionField<T>::zero;
02465     };
02466
02467     template<typename T, size_t i>
02468     struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02469         using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02470     };
02471
02472     template<typename T, size_t i>
02473     struct cos_coeff {
02474         using type = typename cos_coeff_helper<T, i>::type;
02475     };
02476
02477     template<typename T, size_t i, typename E = void>
02478     struct cosh_coeff_helper {};
02479
02480     template<typename T, size_t i>
02481     struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02482         using type = typename FractionField<T>::zero;
02483     };
02484
02485     template<typename T, size_t i>
02486     struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02487         using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02488     };
02489
02490     template<typename T, size_t i>
02491     struct cosh_coeff {
02492         using type = typename cosh_coeff_helper<T, i>::type;
02493     };
02494
02495     template<typename T, size_t i>
02496     struct geom_coeff { using type = typename FractionField<T>::one; };
02497
02498
02499     template<typename T, size_t i, typename E = void>
02500     struct atan_coeff_helper;
02501
02502     template<typename T, size_t i>
02503     struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02504         using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;
02505     };
02506
02507     template<typename T, size_t i>
02508     struct atan_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 atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
02514
02515     template<typename T, size_t i, typename E = void>
02516     struct asin_coeff_helper;
02517
02518     template<typename T, size_t i>
02519     struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02520         using type = makefraction_t<T,
02521             factorial_t<T, i - 1>,
02522             typename T::template mul_t<
02523                 typename T::template val<i>,
02524                 T::template mul_t<
02525                     pow_t<T, 4, i / 2>,
02526                     pow<T, factorial<T, i / 2>::value, 2
02527                 >
02528             >
02529         >>;
02530     };
02531
02532     template<typename T, size_t i>
02533     struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02534         using type = typename FractionField<T>::zero;
02535     };
02536
02537     template<typename T, size_t i>
02538     struct asin_coeff {
02539         using type = typename asin_coeff_helper<T, i>::type;

```

```

02540     };
02541
02542     template<typename T, size_t i>
02543     struct lnpl_coeff {
02544         using type = makefraction_t<T,
02545             alternate_t<T, i + 1>,
02546             typename T::template val<i>;
02547     };
02548
02549     template<typename T>
02550     struct lnpl_coeff<T, 0> { using type = typename FractionField<T>::zero; };
02551
02552     template<typename T, size_t i, typename E = void>
02553     struct asinh_coeff_helper;
02554
02555     template<typename T, size_t i>
02556     struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02557         using type = makefraction_t<T,
02558             typename T::template mul_t<
02559                 alternate_t<T, i / 2>,
02560                 factorial_t<T, i - 1>
02561             >,
02562             typename T::template mul_t<
02563                 typename T::template mul_t<
02564                     typename T::template val<i>,
02565                     pow_t<T, (factorial<T, i / 2>::value), 2>
02566                 >,
02567                 pow_t<T, 4, i / 2>
02568             >
02569         >;
02570     };
02571
02572     template<typename T, size_t i>
02573     struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02574         using type = typename FractionField<T>::zero;
02575     };
02576
02577     template<typename T, size_t i>
02578     struct asinh_coeff {
02579         using type = typename asinh_coeff_helper<T, i>::type;
02580     };
02581
02582     template<typename T, size_t i, typename E = void>
02583     struct atanh_coeff_helper;
02584
02585     template<typename T, size_t i>
02586     struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
02587         // 1/i
02588         using type = typename FractionField<T>::template val<
02589             typename T::one,
02590             typename T::template inject_constant_t<i>;
02591     };
02592
02593     template<typename T, size_t i>
02594     struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0> {
02595         using type = typename FractionField<T>::zero;
02596     };
02597
02598     template<typename T, size_t i>
02599     struct atanh_coeff {
02600         using type = typename atanh_coeff_helper<T, i>::type;
02601     };
02602
02603     template<typename T, size_t i, typename E = void>
02604     struct tan_coeff_helper;
02605
02606     template<typename T, size_t i>
02607     struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0> {
02608         using type = typename FractionField<T>::zero;
02609     };
02610
02611     template<typename T, size_t i>
02612     struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0> {
02613     private:
02614         // 4^((i+1)/2)
02615         using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2>;
02616         // 4^((i+1)/2) - 1
02617         using _4pml = typename FractionField<T>::template sub_t<_4p, typename
FractionField<T>::one>;
02618         // (-1)^((i-1)/2)
02619         using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2>;
02620         using dividend = typename FractionField<T>::template mul_t<
02621             altp,
02622             FractionField<T>::template mul_t<
02623                 _4p,
02624                 FractionField<T>::template mul_t<
02625                     _4pml,

```

```

02626         bernoulli_t<T, (i + 1)>
02627         >
02628         >
02629     >;
02630     public:
02631         using type = typename FractionField<T>::template div_t<dividend,
02632             typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>>;
02633     };
02634
02635     template<typename T, size_t i>
02636     struct tan_coeff {
02637         using type = typename tan_coeff_helper<T, i>::type;
02638     };
02639
02640     template<typename T, size_t i, typename E = void>
02641     struct tanh_coeff_helper;
02642
02643     template<typename T, size_t i>
02644     struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0> {
02645         using type = typename FractionField<T>::zero;
02646     };
02647
02648     template<typename T, size_t i>
02649     struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0> {
02650     private:
02651         using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2>>;
02652         using _4pml = typename FractionField<T>::template sub_t<_4p, typename
FractionField<T>::one>;
02653         using dividend =
02654             typename FractionField<T>::template mul_t<
02655                 _4p,
02656                 typename FractionField<T>::template mul_t<
02657                     _4pml,
02658                     bernoulli_t<T, (i + 1)>>::type>;
02659     public:
02660         using type = typename FractionField<T>::template div_t<dividend,
02661             FractionField<T>::template inject_t<factorial_t<T, i + 1>>>;
02662     };
02663
02664     template<typename T, size_t i>
02665     struct tanh_coeff {
02666         using type = typename tanh_coeff_helper<T, i>::type;
02667     };
02668 } // namespace internal
02669
02670 template<typename Integers, size_t deg>
02671 using exp = taylor<Integers, internal::exp_coeff, deg>;
02672
02673 template<typename Integers, size_t deg>
02674 using expml = typename polynomial<FractionField<Integers>>::template sub_t<
02675     exp<Integers, deg>,
02676     typename polynomial<FractionField<Integers>>::one>;
02677
02678 template<typename Integers, size_t deg>
02679 using lnpl = taylor<Integers, internal::lnpl_coeff, deg>;
02680
02681 template<typename Integers, size_t deg>
02682 using atan = taylor<Integers, internal::atan_coeff, deg>;
02683
02684 template<typename Integers, size_t deg>
02685 using sin = taylor<Integers, internal::sin_coeff, deg>;
02686
02687 template<typename Integers, size_t deg>
02688 using sinh = taylor<Integers, internal::sh_coeff, deg>;
02689
02690 template<typename Integers, size_t deg>
02691 using cosh = taylor<Integers, internal::cosh_coeff, deg>;
02692
02693 template<typename Integers, size_t deg>
02694 using cos = taylor<Integers, internal::cos_coeff, deg>;
02695
02696 template<typename Integers, size_t deg>
02697 using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
02698
02699 template<typename Integers, size_t deg>
02700 using asin = taylor<Integers, internal::asin_coeff, deg>;
02701
02702 template<typename Integers, size_t deg>
02703 using asinh = taylor<Integers, internal::asinh_coeff, deg>;
02704
02705 template<typename Integers, size_t deg>
02706 using atanh = taylor<Integers, internal::atanh_coeff, deg>;
02707
02708 template<typename Integers, size_t deg>
02709 using tan = taylor<Integers, internal::tan_coeff, deg>;
02710
02711 template<typename Integers, size_t deg>

```

```

02762     using tanh = taylor<Integers, internal::tanh_coeff, deg>;
02763 } // namespace aerobus
02764
02765 // continued fractions
02766 namespace aerobus {
02767     template<int64_t... values>
02768     struct ContinuedFraction {};
02769
02770     template<int64_t a0>
02771     struct ContinuedFraction<a0> {
02772         using type = typename q64::template inject_constant_t<a0>;
02773         static constexpr double val = static_cast<double>(a0);
02774     };
02775
02776     template<int64_t a0, int64_t... rest>
02777     struct ContinuedFraction<a0, rest...> {
02778         using type = q64::template add_t<
02779             typename q64::template inject_constant_t<a0>,
02780             typename q64::template div_t<
02781                 typename q64::one,
02782                 typename ContinuedFraction<rest...>::type
02783             >;
02784         static constexpr double val = type::template get<double>();
02785     };
02786
02787     using PI_fraction =
02788     ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1>;
02789     using E_fraction =
02790     ContinuedFraction<2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1>;
02791     using SQRT2_fraction =
02792     ContinuedFraction<1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2>;
02793     using SQRT3_fraction =
02794     ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2>;
02795     // NOLINT
02796 } // namespace aerobus
02797
02798 // known polynomials
02799 namespace aerobus {
02800     // CChebyshev
02801     namespace internal {
02802         template<int kind, size_t deg>
02803         struct chebyshev_helper {
02804             using type = typename pi64::template sub_t<
02805                 typename pi64::template mul_t<
02806                     typename pi64::template mul_t<
02807                         pi64::inject_constant_t<2>,
02808                         typename pi64::X>,
02809                     typename chebyshev_helper<kind, deg - 1>::type
02810                 >,
02811                 typename chebyshev_helper<kind, deg - 2>::type
02812             >;
02813         };
02814
02815         template<>
02816         struct chebyshev_helper<1, 0> {
02817             using type = typename pi64::one;
02818         };
02819
02820         template<>
02821         struct chebyshev_helper<1, 1> {
02822             using type = typename pi64::X;
02823         };
02824
02825         template<>
02826         struct chebyshev_helper<2, 0> {
02827             using type = typename pi64::one;
02828         };
02829
02830         template<>
02831         struct chebyshev_helper<2, 1> {
02832             using type = pi64::template mul_t<
02833                 typename pi64::inject_constant_t<2>,
02834                 typename pi64::X>;
02835         };
02836     } // namespace internal
02837
02838     // Laguerre
02839     namespace internal {
02840         template<size_t deg>
02841         struct laguerre_helper {
02842         private:
02843             // Lk = (1 / k) * ((2 * k - 1 - x) * lkm1 - (k - 2)lkm2)
02844             using lnm2 = typename laguerre_helper<deg - 2>::type;
02845             using lnm1 = typename laguerre_helper<deg - 1>::type;
02846             // -x + 2k-1
02847             using p = typename pq64::template val<
02848                 typename q64::template inject_constant_t<-1>,

```

```

02869         typename pq64::template inject_constant_t<2 * deg - 1>;
02870         // 1/n
02871         using factor = typename pq64::template inject_ring_t<
02872             pq64::val<typename i64::one, typename i64::template inject_constant_t<deg>>>;
02873
02874     public:
02875         using type = typename pq64::template mul_t <
02876             factor,
02877             typename pq64::template sub_t<
02878                 typename pq64::template mul_t<
02879                     p,
02880                     lnm1
02881                 >,
02882                 typename pq64::template mul_t<
02883                     typename pq64::template inject_constant_t<deg-1>,
02884                     lnm2
02885                 >
02886             >
02887         >;
02888     };
02889
02890     template<>
02891     struct laguerre_helper<0> {
02892         using type = typename pq64::one;
02893     };
02894
02895     template<>
02896     struct laguerre_helper<1> {
02897         using type = typename pq64::template sub_t<typename pq64::one, typename pq64::X>;
02898     };
02899 } // namespace internal
02900
02901 // Bernstein
02902 namespace internal {
02903     template<size_t i, size_t m, typename E = void>
02904     struct bernstein_helper {};
02905
02906     template<>
02907     struct bernstein_helper<0, 0> {
02908         using type = typename pi64::one;
02909     };
02910
02911     template<size_t i, size_t m>
02912     struct bernstein_helper<i, m, std::enable_if_t<
02913         (m > 0) && (i == 0)>> {
02914         using type = typename pi64::mul_t<
02915             typename pi64::sub_t<typename pi64::one, typename pi64::X>,
02916             typename bernstein_helper<i, m-1>::type>;
02917     };
02918
02919     template<size_t i, size_t m>
02920     struct bernstein_helper<i, m, std::enable_if_t<
02921         (m > 0) && (i == m)>> {
02922         using type = typename pi64::template mul_t<
02923             typename pi64::X,
02924             typename bernstein_helper<i-1, m-1>::type>;
02925     };
02926
02927     template<size_t i, size_t m>
02928     struct bernstein_helper<i, m, std::enable_if_t<
02929         (m > 0) && (i > 0) && (i < m)>> {
02930         using type = typename pi64::add_t<
02931             typename pi64::mul_t<
02932                 typename pi64::sub_t<typename pi64::one, typename pi64::X>,
02933                 typename bernstein_helper<i, m-1>::type>,
02934             typename pi64::mul_t<
02935                 typename pi64::X,
02936                 typename bernstein_helper<i-1, m-1>::type>;
02937     };
02938 } // namespace internal
02939
02940 namespace known_polynomials {
02941     enum hermite_kind {
02942         probabilist,
02943         physicist
02944     };
02945 }
02946
02947 // hermite
02948 namespace internal {
02949     template<size_t deg, known_polynomials::hermite_kind kind>
02950     struct hermite_helper {};
02951
02952     template<size_t deg>
02953     struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist> {
02954     private:
02955         using hnm1 = typename hermite_helper<deg - 1,

```



```

known_polynomials::hermite_kind::probabilist>::type;
02959     using hnm2 = typename hermite_helper<deg - 2,
known_polynomials::hermite_kind::probabilist>::type;
02960
02961     public:
02962         using type = typename pi64::template sub_t<
02963             typename pi64::template mul_t<typename pi64::X, hnm1>,
02964             typename pi64::template mul_t<
02965                 typename pi64::template inject_constant_t<deg - 1>,
02966                 hnm2
02967             >
02968         >;
02969     };
02970
02971     template<size_t deg>
02972     struct hermite_helper<deg, known_polynomials::hermite_kind::physicist> {
02973     private:
02974         using hnm1 = typename hermite_helper<deg - 1,
known_polynomials::hermite_kind::physicist>::type;
02975         using hnm2 = typename hermite_helper<deg - 2,
known_polynomials::hermite_kind::physicist>::type;
02976
02977     public:
02978         using type = typename pi64::template sub_t<
02979             // 2X Hn-1
02980             typename pi64::template mul_t<
02981                 typename pi64::val<typename i64::template inject_constant_t<2>,
02982                 typename i64::zero>, hnm1>,
02983
02984             typename pi64::template mul_t<
02985                 typename pi64::template inject_constant_t<2*(deg - 1)>,
02986                 hnm2
02987             >
02988         >;
02989     };
02990
02991     template<>
02992     struct hermite_helper<0, known_polynomials::hermite_kind::probabilist> {
02993     private:
02994         using type = typename pi64::one;
02995     };
02996
02997     template<>
02998     struct hermite_helper<1, known_polynomials::hermite_kind::probabilist> {
02999     private:
03000         using type = typename pi64::X;
03001     };
03002
03003     template<>
03004     struct hermite_helper<0, known_polynomials::hermite_kind::physicist> {
03005     private:
03006         using type = typename pi64::one;
03007     };
03008
03009     template<>
03010     struct hermite_helper<1, known_polynomials::hermite_kind::physicist> {
03011     private:
03012         // 2X
03013         using type = typename pi64::template val<typename i64::template inject_constant_t<2>,
03014         typename i64::zero>;
03015     };
03016 } // namespace internal
03017
03018 // legendre
03019 namespace internal {
03020     template<size_t n>
03021     struct legendre_helper {
03022     private:
03023         // 1/n constant
03024         // (2n-1)/n X
03025         using fact_left = typename pq64::monomial_t<make_q64_t<2*n-1, n>, 1>;
03026         // (n-1) / n
03027         using fact_right = typename pq64::val<make_q64_t<n-1, n>;
03028     public:
03029         using type = pq64::template sub_t<
03030             typename pq64::template mul_t<
03031                 fact_left,
03032                 typename legendre_helper<n-1>::type
03033             >,
03034             typename pq64::template mul_t<
03035                 fact_right,
03036                 typename legendre_helper<n-2>::type
03037             >
03038         >;
03039     };
03040
03041     template<>
03042     struct legendre_helper<0> {
03043     private:
03044         using type = typename pq64::one;
03045     };
03046 }

```

```

03041     template<>
03042     struct legendre_helper<1> {
03043         using type = typename pq64::X;
03044     };
03045 } // namespace internal
03046
03047 // bernoulli polynomials
03048 namespace internal {
03049     template<size_t n>
03050     struct bernoulli_coeff {
03051         template<typename T, size_t i>
03052         struct inner {
03053             private:
03054                 using F = FractionField<T>;
03055             public:
03056                 using type = typename F::template mul_t<
03057                     typename F::template inject_ring_t<combination_t<T, i, n>,
03058                         bernoulli_t<T, n-i>
03059                 >;
03060             };
03061         };
03062     } // namespace internal
03063
03064     namespace known_polynomials {
03072         template <size_t deg>
03073         using chebyshev_T = typename internal::chebyshev_helper<1, deg>::type;
03074
03081         template <size_t deg>
03082         using chebyshev_U = typename internal::chebyshev_helper<2, deg>::type;
03083
03090         template <size_t deg>
03091         using laguerre = typename internal::laguerre_helper<deg>::type;
03092
03099         template <size_t deg>
03100         using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist>::type;
03101
03108         template <size_t deg>
03109         using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist>::type;
03110
03118         template<size_t i, size_t m>
03119         using bernstein = typename internal::bernstein_helper<i, m>::type;
03120
03127         template<size_t deg>
03128         using legendre = typename internal::legendre_helper<deg>::type;
03129
03136         template<size_t deg>
03137         using bernoulli = taylor<i64, internal::bernoulli_coeff<deg>::template inner, deg>;
03138     } // namespace known_polynomials
03139 } // namespace aerobus
03140
03141
03142 #ifdef AEROBUS_CONWAY_IMPORTS
03143
03144 // conway polynomials
03145 namespace aerobus {
03149     template<int p, int n>
03150     struct ConwayPolynomial {};
03151
03152 #ifndef DO_NOT_DOCUMENT
03153     #define ZPV ZPZ::template val
03154     #define POLYV aerobus::polynomial<ZPV>::template val
03155     template<> struct ConwayPolynomial<2, 1> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<1>; }; // NOLINT
03156     template<> struct ConwayPolynomial<2, 2> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<1>, ZPV<1>; }; // NOLINT
03157     template<> struct ConwayPolynomial<2, 3> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<1>, ZPV<1>; }; // NOLINT
03158     template<> struct ConwayPolynomial<2, 4> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<1>; }; // NOLINT
03159     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
03160     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
03161     template<> struct ConwayPolynomial<2, 7> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<1>, ZPV<1>; }; // NOLINT
03162     template<> struct ConwayPolynomial<2, 8> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<1>, ZPV<0>, ZPV<1>; }; // NOLINT
03163     template<> struct ConwayPolynomial<2, 9> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<1>; }; //
NOLINT
03164     template<> struct ConwayPolynomial<2, 10> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<1>, ZPV<1>, ZPV<0>, ZPV<1>, ZPV<1>, ZPV<1>,
ZPV<1>; }; // NOLINT
03165     template<> struct ConwayPolynomial<2, 11> { using ZPV = aerobus::zpv<2>; using type =
POLYV<ZPV<1>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<0>, ZPV<1>,
ZPV<0>, ZPV<1>; }; // NOLINT
03166     template<> struct ConwayPolynomial<2, 12> { 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

[illegible]

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

```

05070     template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5>; }; // NOLINT
05071     template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978>; }; // NOLINT
05072     template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5>; }; // NOLINT
05073     template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<978>; }; // NOLINT
05074     template<> struct ConwayPolynomial<983, 6> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<849>, ZPZV<296>, ZPZV<228>, ZPZV<5>; }; // NOLINT
05075     template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<978>; }; // NOLINT
05076     template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<5>; }; //
NOLINT
05077     template<> struct ConwayPolynomial<983, 9> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<858>, ZPZV<87>, ZPZV<978>;
}; // NOLINT
05078     template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<985>; }; // NOLINT
05079     template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6>; }; // NOLINT
05080     template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985>; }; // NOLINT
05081     template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6>; }; // NOLINT
05082     template<> struct ConwayPolynomial<991, 5> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<985>; }; // NOLINT
05083     template<> struct ConwayPolynomial<991, 6> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<637>, ZPZV<855>, ZPZV<278>, ZPZV<6>; }; // NOLINT
05084     template<> struct ConwayPolynomial<991, 7> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<985>; }; // NOLINT
05085     template<> struct ConwayPolynomial<991, 8> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<941>, ZPZV<786>, ZPZV<234>, ZPZV<6>; }; //
NOLINT
05086     template<> struct ConwayPolynomial<991, 9> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<466>, ZPZV<222>, ZPZV<985>;
}; // NOLINT
05087     template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<990>; }; // NOLINT
05088     template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7>; }; // NOLINT
05089     template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990>; }; // NOLINT
05090     template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7>; }; // NOLINT
05091     template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<990>; }; // NOLINT
05092     template<> struct ConwayPolynomial<997, 6> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<981>, ZPZV<58>, ZPZV<260>, ZPZV<7>; }; // NOLINT
05093     template<> struct ConwayPolynomial<997, 7> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<990>; }; // NOLINT
05094     template<> struct ConwayPolynomial<997, 8> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<934>, ZPZV<473>, ZPZV<241>, ZPZV<7>; }; //
NOLINT
05095     template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<732>, ZPZV<616>, ZPZV<990>;
}; // NOLINT
05096 #endif // DO_NOT_DOCUMENT
05097 } // namespace aerobus
05098 #endif // AEROBUS_CONWAY_IMPORTS
05099
05100 #endif // __INC_AEROBUS__ // NOLINT

```


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><i32::val<1>></code> |
|-----|---|

10.14 i64::template

injects constant as an i64 value

injects constant as an i64 value

Template Parameters

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

10.15 i64::add_t

addition operator

addition operator

Template Parameters

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

10.16 i64::sub_t

subtraction operator

subtraction operator

Template Parameters

| | |
|----|--|
| v1 | : an element of aerobus::i64::val |
| v2 | : an element of aerobus::i64::val <i64::val<1>, i64::val<2>> |

10.17 i64::mul_t

multiplication operator

multiplication operator

Template Parameters

| | |
|----|--|
| v1 | : an element of aerobus::i64::val |
| v2 | : an element of aerobus::i64::val <i64::val<1>, i64::val<2>> |

10.18 i64::div_t

division operator integer division

division operator integer division

Template Parameters

| | |
|----|--|
| v1 | : an element of aerobus::i64::val |
| v2 | : an element of aerobus::i64::val <i64::val<1>, i64::val<2>> |

10.19 i64::mod_t

modulus operator

modulus operator

Template Parameters

| | |
|----|---|
| v1 | : an element of aerobus::i64::val |
| v2 | : an element of aerobus::i64::val <i64::val<6>, i64::val<15>> |

10.20 i64::gt_t

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

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

Template Parameters

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

10.21 i64::lt_t

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

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

Template Parameters

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

10.22 i64::lt_v

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

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

Template Parameters

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

10.23 i64::eq_t

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

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

Template Parameters

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

10.24 i64::eq_v

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

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

Template Parameters

| | |
|-----------------|---|
| <code>v1</code> | : an element of aerobus::i64::val |
| <code>v2</code> | : an element of aerobus::i64::val <code><i64::val<2>, i64::val<2>></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 aerobus::i64::val |
| <code>v2</code> | : an element of aerobus::i64::val <code><i64::val<6>, i64::val<15>></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>v</code> | : an element of aerobus::i64::val <code><i64::val<1>></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 aerobus::i64::val <code><i64::val<1>></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><i32>::template inject_constant_t<2></code> |
|----------------|---|

10.29 q32::add_t

addition operator

addition operator

Template Parameters

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

10.30 FractionField

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

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

Template Parameters

| | |
|-------------------|--|
| <code>Ring</code> | <code><i64></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](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 π 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, 19

add_t
 aerobus::i32, 46
 aerobus::i64, 49
 aerobus::polynomial< Ring >, 55
 aerobus::Quotient< Ring, X >, 62
 aerobus::zpz< p >, 83

addfractions_t
 aerobus, 19

aerobus, 15
 abs_t, 19
 addfractions_t, 19
 aligned_malloc, 28
 alternate_t, 19
 alternate_v, 29
 asin, 19
 asinh, 20
 atan, 20
 atanh, 20
 bernoulli_t, 20
 bernoulli_v, 29
 combination_t, 21
 combination_v, 29
 cos, 21
 cosh, 21
 E_fraction, 21
 exp, 22
 expm1, 22
 factorial_t, 22
 factorial_v, 30
 field, 29
 fpq32, 22
 fpq64, 22
 FractionField, 23
 gcd_t, 23
 geometric_sum, 23
 lnp1, 23
 make_q32_t, 24
 make_q64_t, 24
 makefraction_t, 24
 mulfractions_t, 24
 pi64, 25
 PI_fraction, 25
 pow_t, 25
 pq64, 25
 q32, 25
 q64, 25
 sin, 26
 sinh, 26
 SQRT2_fraction, 26
 SQRT3_fraction, 26
 stirling_signed_t, 26
 stirling_unsigned_t, 27
 tan, 27
 tanh, 27
 taylor, 27
 vadd_t, 28
 vmul_t, 28

aerobus::ContinuedFraction< a0 >, 42
 type, 43
 val, 43

aerobus::ContinuedFraction< a0, rest... >, 43
 type, 44
 val, 44

aerobus::ContinuedFraction< values >, 42

aerobus::ConwayPolynomial, 44

aerobus::i32, 44
 add_t, 46
 div_t, 46
 eq_t, 46
 eq_v, 47
 gcd_t, 46
 gt_t, 46
 inject_constant_t, 46
 inject_ring_t, 46
 inner_type, 46
 is_euclidean_domain, 47
 is_field, 47
 lt_t, 46
 mod_t, 46
 mul_t, 47
 one, 47
 pos_t, 47
 pos_v, 48
 sub_t, 47
 zero, 47

aerobus::i32::val< x >, 70
 enclosing_type, 71
 eval, 71
 get, 71
 is_zero_t, 71
 to_string, 72
 v, 72

aerobus::i64, 48
 add_t, 49
 div_t, 49
 eq_t, 49

- eq_v, 51
- gcd_t, 49
- gt_t, 50
- gt_v, 51
- inject_constant_t, 50
- inject_ring_t, 50
- inner_type, 50
- is_euclidean_domain, 52
- is_field, 52
- lt_t, 50
- lt_v, 52
- mod_t, 50
- mul_t, 50
- one, 51
- pos_t, 51
- pos_v, 52
- sub_t, 51
- zero, 51
- aerobus::i64::val< x >, 72
 - enclosing_type, 73
 - eval, 73
 - get, 73
 - is_zero_t, 73
 - to_string, 74
 - v, 74
- aerobus::internal, 30
 - index_sequence_reverse, 34
 - is_instantiation_of_v, 34
 - make_index_sequence_reverse, 33
 - type_at_t, 33
- aerobus::is_prime< n >, 52
 - value, 53
- aerobus::IsEuclideanDomain, 39
- aerobus::IsField, 39
- aerobus::IsRing, 40
- aerobus::known_polynomials, 34
 - bernoulli, 35
 - bernstein, 35
 - chebyshev_T, 35
 - chebyshev_U, 36
 - hermite_kind, 38
 - hermite_phys, 36
 - hermite_prob, 36
 - laguerre, 37
 - legendre, 37
 - physicist, 38
 - probabilist, 38
- aerobus::polynomial< Ring >, 53
 - add_t, 55
 - derive_t, 55
 - div_t, 55
 - eq_t, 55
 - gcd_t, 56
 - gt_t, 56
 - inject_constant_t, 56
 - inject_ring_t, 56
 - is_euclidean_domain, 59
 - is_field, 59
 - lt_t, 56
 - mod_t, 57
 - monomial_t, 57
 - mul_t, 57
 - one, 58
 - pos_t, 58
 - pos_v, 59
 - simplify_t, 58
 - sub_t, 58
 - X, 59
 - zero, 59
- aerobus::polynomial< Ring >::val< coeffN >, 79
 - aN, 80
 - coeff_at_t, 80
 - degree, 82
 - enclosing_type, 81
 - eval, 81
 - is_zero_t, 81
 - is_zero_v, 82
 - strip, 81
 - to_string, 81
- aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E >, 41
- aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0 || index > 0)> >, 41
 - type, 41
- aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> >, 42
 - type, 42
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 74
 - aN, 75
 - coeff_at_t, 75
 - degree, 77
 - enclosing_type, 75
 - eval, 76
 - is_zero_t, 76
 - is_zero_v, 77
 - strip, 76
 - to_string, 76
- aerobus::Quotient< Ring, X >, 60
 - add_t, 62
 - div_t, 62
 - eq_t, 62
 - eq_v, 64
 - inject_constant_t, 62
 - inject_ring_t, 63
 - is_euclidean_domain, 64
 - mod_t, 63
 - mul_t, 63
 - one, 63
 - pos_t, 63
 - pos_v, 64
 - zero, 64
- aerobus::Quotient< Ring, X >::val< V >, 77
 - type, 78
- aerobus::type_list< Ts >, 66

- at, 67
- concat, 67
- insert, 67
- length, 68
- push_back, 67
- push_front, 68
- remove, 68
- aerobus::type_list< Ts >::pop_front, 59
- tail, 60
- type, 60
- aerobus::type_list< Ts >::split< index >, 65
- head, 65
- tail, 65
- aerobus::type_list<>, 68
- concat, 69
- insert, 69
- length, 70
- push_back, 69
- push_front, 69
- aerobus::zpz< p >, 82
- add_t, 83
- div_t, 84
- eq_t, 84
- eq_v, 87
- gcd_t, 84
- gt_t, 84
- gt_v, 87
- inject_constant_t, 85
- inner_type, 85
- is_euclidean_domain, 87
- is_field, 87
- lt_t, 85
- lt_v, 87
- mod_t, 85
- mul_t, 86
- one, 86
- pos_t, 86
- pos_v, 88
- sub_t, 86
- zero, 87
- aerobus::zpz< p >::val< x >, 78
- enclosing_type, 78
- eval, 79
- get, 79
- is_zero_t, 78
- to_string, 79
- v, 79
- aligned_malloc
- aerobus, 28
- alternate_t
- aerobus, 19
- alternate_v
- aerobus, 29
- aN
- aerobus::polynomial< Ring >::val< coeffN >, 80
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 75
- asin
- aerobus, 19
- asinh
- aerobus, 20
- at
- aerobus::type_list< Ts >, 67
- atan
- aerobus, 20
- atanh
- aerobus, 20
- bernoulli
- aerobus::known_polynomials, 35
- bernoulli_t
- aerobus, 20
- bernoulli_v
- aerobus, 29
- bernstein
- aerobus::known_polynomials, 35
- chebyshev_T
- aerobus::known_polynomials, 35
- chebyshev_U
- aerobus::known_polynomials, 36
- coeff_at_t
- aerobus::polynomial< Ring >::val< coeffN >, 80
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 75
- combination_t
- aerobus, 21
- combination_v
- aerobus, 29
- concat
- aerobus::type_list< Ts >, 67
- aerobus::type_list<>, 69
- cos
- aerobus, 21
- cosh
- aerobus, 21
- degree
- aerobus::polynomial< Ring >::val< coeffN >, 82
- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 77
- derive_t
- aerobus::polynomial< Ring >, 55
- div_t
- aerobus::i32, 46
- aerobus::i64, 49
- aerobus::polynomial< Ring >, 55
- aerobus::Quotient< Ring, X >, 62
- aerobus::zpz< p >, 84
- E_fraction
- aerobus, 21
- enclosing_type
- aerobus::i32::val< x >, 71
- aerobus::i64::val< x >, 73
- aerobus::polynomial< Ring >::val< coeffN >, 81

- aerobus::polynomial< Ring >::val< coeffN, coeffs >, 75
- aerobus::zpz< p >::val< x >, 78
- eq_t
 - aerobus::i32, 46
 - aerobus::i64, 49
 - aerobus::polynomial< Ring >, 55
 - aerobus::Quotient< Ring, X >, 62
 - aerobus::zpz< p >, 84
- eq_v
 - aerobus::i32, 47
 - aerobus::i64, 51
 - aerobus::Quotient< Ring, X >, 64
 - aerobus::zpz< p >, 87
- eval
 - aerobus::i32::val< x >, 71
 - aerobus::i64::val< x >, 73
 - aerobus::polynomial< Ring >::val< coeffN >, 81
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 76
 - aerobus::zpz< p >::val< x >, 79
- exp
 - aerobus, 22
- expm1
 - aerobus, 22
- factorial_t
 - aerobus, 22
- factorial_v
 - aerobus, 30
- field
 - aerobus, 29
- fpq32
 - aerobus, 22
- fpq64
 - aerobus, 22
- FractionField
 - aerobus, 23
- gcd_t
 - aerobus, 23
 - aerobus::i32, 46
 - aerobus::i64, 49
 - aerobus::polynomial< Ring >, 56
 - aerobus::zpz< p >, 84
- geometric_sum
 - aerobus, 23
- get
 - aerobus::i32::val< x >, 71
 - aerobus::i64::val< x >, 73
 - aerobus::zpz< p >::val< x >, 79
- gt_t
 - aerobus::i32, 46
 - aerobus::i64, 50
 - aerobus::polynomial< Ring >, 56
 - aerobus::zpz< p >, 84
- gt_v
 - aerobus::i64, 51
 - aerobus::zpz< p >, 87
- head
 - aerobus::type_list< Ts >::split< index >, 65
- hermite_kind
 - aerobus::known_polynomials, 38
- hermite_phys
 - aerobus::known_polynomials, 36
- hermite_prob
 - aerobus::known_polynomials, 36
- index_sequence_reverse
 - aerobus::internal, 34
- inject_constant_t
 - aerobus::i32, 46
 - aerobus::i64, 50
 - aerobus::polynomial< Ring >, 56
 - aerobus::Quotient< Ring, X >, 62
 - aerobus::zpz< p >, 85
- inject_ring_t
 - aerobus::i32, 46
 - aerobus::i64, 50
 - aerobus::polynomial< Ring >, 56
 - aerobus::Quotient< Ring, X >, 63
- inner_type
 - aerobus::i32, 46
 - aerobus::i64, 50
 - aerobus::zpz< p >, 85
- insert
 - aerobus::type_list< Ts >, 67
 - aerobus::type_list<>, 69
- Introduction, 1
- is_euclidean_domain
 - aerobus::i32, 47
 - aerobus::i64, 52
 - aerobus::polynomial< Ring >, 59
 - aerobus::Quotient< Ring, X >, 64
 - aerobus::zpz< p >, 87
- is_field
 - aerobus::i32, 47
 - aerobus::i64, 52
 - aerobus::polynomial< Ring >, 59
 - aerobus::zpz< p >, 87
- is_instantiation_of_v
 - aerobus::internal, 34
- is_zero_t
 - aerobus::i32::val< x >, 71
 - aerobus::i64::val< x >, 73
 - aerobus::polynomial< Ring >::val< coeffN >, 81
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 76
 - aerobus::zpz< p >::val< x >, 78
- is_zero_v
 - aerobus::polynomial< Ring >::val< coeffN >, 82
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 77
- laguerre
 - aerobus::known_polynomials, 37
- legendre
 - aerobus::known_polynomials, 37

- length
 - aerobus::type_list< Ts >, 68
 - aerobus::type_list<>, 70
- lnp1
 - aerobus, 23
- lt_t
 - aerobus::i32, 46
 - aerobus::i64, 50
 - aerobus::polynomial< Ring >, 56
 - aerobus::zpz< p >, 85
- lt_v
 - aerobus::i64, 52
 - aerobus::zpz< p >, 87
- main_page.md, 89
- make_index_sequence_reverse
 - aerobus::internal, 33
- make_q32_t
 - aerobus, 24
- make_q64_t
 - aerobus, 24
- makefraction_t
 - aerobus, 24
- mod_t
 - aerobus::i32, 46
 - aerobus::i64, 50
 - aerobus::polynomial< Ring >, 57
 - aerobus::Quotient< Ring, X >, 63
 - aerobus::zpz< p >, 85
- monomial_t
 - aerobus::polynomial< Ring >, 57
- mul_t
 - aerobus::i32, 47
 - aerobus::i64, 50
 - aerobus::polynomial< Ring >, 57
 - aerobus::Quotient< Ring, X >, 63
 - aerobus::zpz< p >, 86
- mulfractions_t
 - aerobus, 24
- one
 - aerobus::i32, 47
 - aerobus::i64, 51
 - aerobus::polynomial< Ring >, 58
 - aerobus::Quotient< Ring, X >, 63
 - aerobus::zpz< p >, 86
- physicist
 - aerobus::known_polynomials, 38
- pi64
 - aerobus, 25
- PI_fraction
 - aerobus, 25
- pos_t
 - aerobus::i32, 47
 - aerobus::i64, 51
 - aerobus::polynomial< Ring >, 58
 - aerobus::Quotient< Ring, X >, 63
 - aerobus::zpz< p >, 86
- pos_v
 - aerobus::i32, 48
 - aerobus::i64, 52
 - aerobus::polynomial< Ring >, 59
 - aerobus::Quotient< Ring, X >, 64
 - aerobus::zpz< p >, 88
- pow_t
 - aerobus, 25
- pq64
 - aerobus, 25
- probabilist
 - aerobus::known_polynomials, 38
- push_back
 - aerobus::type_list< Ts >, 67
 - aerobus::type_list<>, 69
- push_front
 - aerobus::type_list< Ts >, 68
 - aerobus::type_list<>, 69
- q32
 - aerobus, 25
- q64
 - aerobus, 25
- remove
 - aerobus::type_list< Ts >, 68
- simplify_t
 - aerobus::polynomial< Ring >, 58
- sin
 - aerobus, 26
- sinh
 - aerobus, 26
- SQRT2_fraction
 - aerobus, 26
- SQRT3_fraction
 - aerobus, 26
- src/aerobus.h, 89
- stirling_signed_t
 - aerobus, 26
- stirling_unsigned_t
 - aerobus, 27
- strip
 - aerobus::polynomial< Ring >::val< coeffN >, 81
 - aerobus::polynomial< Ring >::val< coeffN, coeffs >, 76
- sub_t
 - aerobus::i32, 47
 - aerobus::i64, 51
 - aerobus::polynomial< Ring >, 58
 - aerobus::zpz< p >, 86
- tail
 - aerobus::type_list< Ts >::pop_front, 60
 - aerobus::type_list< Ts >::split< index >, 65
- tan
 - aerobus, 27
- tanh
 - aerobus, 27

taylor
 aerobus, 27
 to_string
 aerobus::i32::val< x >, 72
 aerobus::i64::val< x >, 74
 aerobus::polynomial< Ring >::val< coeffN >, 81
 aerobus::polynomial< Ring >::val< coeffN, coeffs
 >, 76
 aerobus::zpz< p >::val< x >, 79
 type
 aerobus::ContinuedFraction< a0 >, 43
 aerobus::ContinuedFraction< a0, rest... >, 44
 aerobus::polynomial< Ring >::val< coeffN
 >::coeff_at< index, std::enable_if_t<(index<
 0 || index > 0)>>, 41
 aerobus::polynomial< Ring >::val< coeffN
 >::coeff_at< index, std::enable_if_t<(index==0)>
 >, 42
 aerobus::Quotient< Ring, X >::val< V >, 78
 aerobus::type_list< Ts >::pop_front, 60
 type_at_t
 aerobus::internal, 33

 v
 aerobus::i32::val< x >, 72
 aerobus::i64::val< x >, 74
 aerobus::zpz< p >::val< x >, 79
 vadd_t
 aerobus, 28
 val
 aerobus::ContinuedFraction< a0 >, 43
 aerobus::ContinuedFraction< a0, rest... >, 44
 value
 aerobus::is_prime< n >, 53
 vmul_t
 aerobus, 28

 X
 aerobus::polynomial< Ring >, 59

 zero
 aerobus::i32, 47
 aerobus::i64, 51
 aerobus::polynomial< Ring >, 59
 aerobus::Quotient< Ring, X >, 64
 aerobus::zpz< p >, 87