Aerobus

v1.2

Generated by Doxygen 1.9.8

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

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

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

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

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

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

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

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

	8.23.2.7 inner_type	87
	8.23.2.8 lt_t	87
	8.23.2.9 mod_t	88
	8.23.2.10 mul_t	88
	8.23.2.11 one	88
	8.23.2.12 pos_t	88
	8.23.2.13 sub_t	89
	8.23.2.14 zero	89
	8.23.3 Member Data Documentation	89
	8.23.3.1 eq_v	89
	8.23.3.2 gt_v	89
	8.23.3.3 is_euclidean_domain	90
	8.23.3.4 is_field	90
	8.23.3.5 lt_v	90
	8.23.3.6 pos_v	90
0	File Documentation	91
9	9.1 README.md File Reference	
	9.2 src/aerobus.h File Reference	
	9.3 aerobus.h	
	3.5 delobus.ii	31
10) Examples	177
	10.1 QuotientRing	177
	10.2 type_list	
	10.3 i32::template	
	10.4 i32::add_t	
	10.5 i32::sub_t	178
	10.6 i32::mul_t	
	10.7 i32::div_t	
	10.8 i32::gt_t	
	10.9 i32::eq_t	
	10.10 i32::eq_v	
	10.11 i32::gcd_t	
	10.12 i32::pos_t	
	10.13 i32::pos_v	
	10.14 i64::template	
	10.15 i64::add_t	
	10.16 i64::sub_t	
	10.17 i64::mul_t	181
		181
	10.19 i64::mod_t	
	10.20 i64::gt_t	
	10.21 i64::lt_t	182

10.22 i64::lt_v	182
10.23 i64::eq_t	182
10.24 i64::eq_v	183
10.25 i64::gcd_t	183
10.26 i64::pos_t	183
10.27 i64::pos_v	183
10.28 polynomial	184
10.29 q32::add_t	184
10.30 FractionField	184
10.31 aerobus::ContinuedFraction	184
10.32 PI_fraction::val	185
10.33 E_fraction::val	185
Index	187

Introduction

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

Everything in Aerobus is expressed as types.

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

Everything is expressed as types

The library serves two main purposes:

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

It is designed to be 'quite easily' extensible.

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

1.1 HOW TO

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

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

2 Introduction

1.1.1 Unit Test

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

Move to the top directory then:

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

Terminal should write:

100% tests passed, 0 tests failed out of 48

Alternate way:

make tests

From top directory.

1.1.2 Benchmarks

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

In addition of Cmake and compiler, install OpenMP. Then move to top directory:

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

results on my laptop:

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

1.2 Structures 3

1.2 Structures

1.2.1 Predefined discrete euclidean domains

Aerobus predefines several simple euclidean domains, such as :

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

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

```
• add_t : a type (specialization of val), representing addition between two values
```

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

and the following "elements":

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

1.2.2 Polynomials

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

```
In addition, values have an evaluation function:
```

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

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

4 Introduction

1.2.3 Known polynomials

Aerobus predefines some well known families of polynomials, such as Hermite or Bernstein: using B23 = aerobus::known_polynomials::bernstein<2, 3>; // $3X^2(1-X)$ constexpr float x = B32::eval(2.0F); // -12

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

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

1.2.4 Conway polynomials

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

```
They can be used to construct finite fields of order p^n ( \mathbb{F}_{p^n}): using F2 = zpz<2>; using PF2 = polynomial<F2>; using F4 = Quotient<PF2, ConwayPolynomial<2, 2>::type>;
```

1.2.5 Taylor series

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

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

1.2 Structures 5

- 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_serie = taylor<Integers, my_coeff_at, degree>;

static constexpr double x = my_serie<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_expm1(const size_t N, double* in, double* out) {
   using V = aerobus::expm1<aerobus::i64, 13>;
   for (size_t i = 0; i < N; ++i) {
      out[i] = V::eval(in[i]);
   }
}</pre>
```

Yields this assembly (clang 17, -mavx2 -03) 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*):
          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]
                  ymm11, QWORD PTR .LC7[rip]
 vbroadcastsd
          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]
                   ymm6, QWORD PTR .LC17[rip]
  vbroadcastsd
                   ymm5, QWORD PTR .LC19[rip]
 vbroadcastsd
  vbroadcastsd
                   ymm4, QWORD PTR .LC21[rip]
```

6 Introduction

```
ymm3, QWORD PTR .LC23[rip]
 vbroadcastsd
                 ymm2, QWORD PTR .LC25[rip]
 vbroadcastsd
.L3:
 vmovupd ymm15, YMMWORD PTR [rsi+rax]
 vmovapd ymm0, ymm15
                 ymm0, ymm14, ymm1
 vfmadd132pd
 vfmadd132pd
                 ymm0, ymm13, ymm15
 vfmadd132pd
                 ymm0, ymm12, ymm15
 vfmadd132pd
                 ymm0, ymm11, ymm15
 vfmadd132pd
                 ymm0, ymm10, ymm15
 vfmadd132pd
                ymm0, ymm9, ymm15
 vfmadd132pd
                 ymm0, ymm8, ymm15
 vfmadd132pd
                 ymm0, ymm7, ymm15
 vfmadd132pd
                 ymm0, ymm6, ymm15
 vfmadd132pd
                 ymm0, ymm5, ymm15
 vfmadd132pd
                 ymm0, ymm4, ymm15
 vfmadd132pd
                 ymm0, ymm3, ymm15
 vfmadd132pd
                 ymm0, ymm2, ymm15
 vfmadd132pd
                 ymm0, ymm1, ymm15
 vmovupd YMMWORD PTR [rdx+rax], ymm0
         rax, 32
 cmp
         rcx, rax
         .L3
 ine
 mov
         rax, rdi
 and
         rax, -4
 vzeroupper
```

1.3 Operations

1.3.1 Field of fractions

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

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

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

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

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

```
For example, we can easily define rational functions by taking the ring of fractions of polynomials: using namespace aerobus; using RF64 = FractionField<polynomial<q64>>;
```

Which also have an evaluation function, as polynomial do.

1.3.2 Quotient

Given a ring R, Aerobus provides automatic implementation for $\ \, \text{quotient ring } R/X \ \, \text{where X is a principal}$ 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 \text{ZpZ} = \text{Quotient} < \text{i64}, i64::val<17>>;
```

As we could have using zpz<17>.

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

1.4 Misc 7

1.4 Misc

1.4.1 Continued Fractions

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

As practical examples, <code>aerobus</code> gives continued fractions of π , e, $\sqrt{2}$ and $\sqrt{3}$: <code>constexpr double A_SQRT3 = aerobus::SQRT3_fraction::val; // 1.7320508075688772935</code>

8 Introduction

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

aerobus	
Main namespace for all publicly exposed types or functions	17
aerobus::internal	
Internal implementations, subject to breaking changes without notice	33
aerobus::known_polynomials	
Families of well known polynomials such as Hermite or Bernstein	36

10 Namespace Index

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	41
aerobus::IsField	
Concept to express R is a field	41
aerobus::IsRing	
Concept to express R is a Ring	42

12 Concept Index

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 >	43
$aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable_if_t < (index < 0 index > 0) > > 43$	
aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)>>	44
aerobus::ContinuedFraction < values >	44
aerobus::ContinuedFraction < a0 >	
Specialization for only one coefficient, technically just 'a0'	44
aerobus::ContinuedFraction < a0, rest >	
Specialization for multiple coefficients (strictly more than one)	45
aerobus::ConwayPolynomial	46
aerobus::i32	
32 bits signed integers, seen as a algebraic ring with related operations	46
aerobus::i64	
64 bits signed integers, seen as a algebraic ring with related operations	50
aerobus::is_prime< n >	
Checks if n is prime	54
aerobus::polynomial < Ring >	55
aerobus::type_list< Ts >::pop_front	
Removes types from head of the list	61
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 Z/2Z	62
aerobus::type_list< Ts >::split< index >	
Splits list at index	67
aerobus::type_list< Ts >	
Empty pure template struct to handle type list	68
aerobus::type_list<>	
Specialization for empty type list	70
aerobus::i32::val < x >	
Values in i32, again represented as types	72
aerobus::i64::val< x >	
Values in i64	74
aerobus::polynomial< Ring >::val< coeffN, coeffs >	
Values (seen as types) in polynomial ring	76
aerobus::Quotient< Ring, X >::val< V >	
Projection values in the quotient ring	79

aerobus::zpz::val< x >	80
aerobus::polynomial < Ring >::val < coeffN >	
Specialization for constants	82
aerobus::zpz	84

File Index

5.1	File	List
-----	------	------

Here is a list of all files	with	brie	f de	scrip	otio	ns:													
src/aerobus.h													 				 		91

16 File Index

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 Z/2Z.

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 or A and B
template<typename... vals>
  using vadd_t = typename internal::vadd< vals... >::type
      adds multiple values (v1 + v2 + ... + vn) vals must have same "enclosing_type" and "enclosing_type" must have an
     add_t binary operator

    template<typename... vals>

  using vmul_t = typename internal::vmul < vals... >::type
      multiplies multiple values (v1 + v2 + ... + vn) vals must have same "enclosing_type" and "enclosing_type" must have
     an mul_t binary operator

    template<typename val >

  using abs t = std::conditional t < val::enclosing type::template pos v < val >, val, typename val::enclosing ←
  _type::template sub_t< typename val::enclosing_type::zero, val > >
      computes absolute value of 'val' val must be a 'value' in a Ring satisfying 'IsEuclideanDomain' concept
• template<typename Ring >
  using FractionField = typename internal::FractionFieldImpl< Ring >::type
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 >>>
```

using make_q32_t = typename q32::template simplify_t< typename q32::val< i32::inject_constant_t< p>,

helper type: make a fraction from numerator and denominator

helper type: make a fraction from numerator and denominator

• template<int32_t p, int32_t q>

 $i32::inject_constant_t < q > > >$

```
• template<typename Ring , typename v1 , typename v2 >
  using addfractions_t = typename FractionField< Ring >::template add t< v1, v2 >
     helper type : adds two fractions

    template<typename Ring , typename v1 , typename v2 >

  using mulfractions t = typename FractionField < Ring >::template mul t < v1, v2 >
     helper type: multiplies two fractions
• template<typename T , size_t i>
  using factorial_t = typename internal::factorial < T, i >::type
     computes factorial(i), as type
• template<typename T , size_t k, size_t n>
  using combination_t = typename internal::combination< T, k, n >::type
     computes binomial coefficient (k among n) as type
template<typename T, size_t n>
  using bernoulli t = typename internal::bernoulli < T, n >::type
      nth bernoulli number as type in T
• template<typename T , size_t n>
  using bell_t = typename internal::bell_helper< T, n >::type
     Bell numbers.

    template<typename T , int k>

  using alternate t = typename internal::alternate < T, k >::type
      (-1)^{\wedge}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 king (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 king (unsigned) - as types.
• template<typename T , typename p , size_t n>
  using pow_t = typename internal::pow< T, p, n >::type
     p^{\wedge}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 >

    template<typename Integers, size t deg>

  using expm1 = typename polynomial < FractionField < Integers > >::template sub_t < exp < Integers, deg
  >, typename polynomial< FractionField< Integers > >::one >
      e^{x} - 1
• template<typename Integers , size_t deg>
  using lnp1 = taylor < Integers, internal::lnp1 coeff, deg >
     ln(1+x)
• template<typename Integers , size_t deg>
  using atan = taylor< Integers, internal::atan_coeff, deg >

    template<typename Integers , size_t deg>

  using sin = taylor< Integers, internal::sin_coeff, deg >
• template<typename Integers , size_t deg>
  using sinh = taylor < Integers, internal::sh_coeff, deg >
     sinh(x)

    template<typename Integers , size_t deg>

  using cosh = taylor < Integers, internal::cosh_coeff, deg >
```

```
\cosh(x) hyperbolic cosine
• template<typename Integers , size_t deg>
     using cos = taylor< Integers, internal::cos_coeff, deg >
             cos(x) cosinus
• template<typename Integers , size_t deg>
     using geometric_sum = taylor< Integers, internal::geom_coeff, deg >
              \frac{1}{1-x} zero development of \frac{1}{1-x}
• template<typename Integers , size_t deg>
     using asin = taylor < Integers, internal::asin coeff, deg >
             \arcsin(x) arc sinus
• template<typename Integers , size_t deg>
     using asinh = taylor < Integers, internal::asinh coeff, deg >
             \operatorname{arcsinh}(x) arc hyperbolic sinus
• template<typename Integers, size t deg>
     using atanh = taylor < Integers, internal::atanh coeff, deg >
             \operatorname{arctanh}(x) arc hyperbolic tangent
• template<typename Integers , size_t deg>
     using tan = taylor< Integers, internal::tan_coeff, deg >
             tan(x) tangent
• template<typename Integers , size_t deg>
     using tanh = taylor < Integers, internal::tanh_coeff, deg >
             tanh(x) hyperbolic tangent

    using PI_fraction = ContinuedFraction < 3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1 >

• using E_fraction = ContinuedFraction < 2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1 >
approximation of \sqrt{2}

    using SQRT3 fraction = ContinuedFraction < 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1
     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

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

```
val a value in a RIng, such as i64::val<-2>
```

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

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

```
T Ring type, aerobus::i64 for example
```

6.1.2.4 asin

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

```
using aerobus::asin = typedef taylor<Integers, internal::asin_coeff, deg> \arcsin(x) arc sinus
```

Template Parameters

Integers	Ring type (for example i64)
deg	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

Integers	Ring type (for example i64)
deg	taylor approximation degree

6.1.2.6 atan

```
template<typename Integers , size_t deg> using aerobus::atan = typedef taylor<Integers, internal::atan_coeff, deg> \arctan(x)
```

Template Parameters

Integers	Ring type (for example i64)
deg	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

Integers	Ring type (for example i64)
deg	taylor approximation degree

6.1.2.8 bell_t

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

Bell numbers.

Template Parameters

T	ring type, such as aerobus::i64
n	index

6.1.2.9 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

T	Ring type (i64)
n	

6.1.2.10 combination_t

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

computes binomial coefficient (k among n) as type

Template Parameters

```
T Ring type (i32 for example)
```

6.1.2.11 cos

```
template<typename Integers , size_t deg> using aerobus::cos = typedef taylor<Integers, internal::cos_coeff, deg> \cos(x) \cos us
```

Template Parameters

Integers	Ring type (for example i64)
deg	taylor approximation degree

6.1.2.12 cosh

```
template<typename Integers , size_t deg> using aerobus::cosh = typedef taylor<Integers, internal::cosh_coeff, deg> \cosh(x) \; \text{hyperbolic cosine}
```

Template Parameters

Integers	Ring type (for example i64)
deg	taylor approximation degree

6.1.2.13 **E_fraction**

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

6.1.2.14 exp

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

Template Parameters

Integers	Ring type (for example i64)
deg	taylor approximation degree

6.1.2.15 expm1

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

$e^x - 1$

Template Parameters

T	Ring type (for example i64)
deg	taylor approximation degree

6.1.2.16 factorial_t

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

computes factorial(i), as type

Template Parameters

T	Ring type (e.g. i32)
i	

6.1.2.17 fpq32

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

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

6.1.2.18 fpq64

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

polynomial with 64 bits rational coefficients

6.1.2.19 FractionField

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

6.1.2.20 gcd_t

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

computes the greatest common divisor or A and B

Template Parameters

```
T Ring type (must be euclidean domain)
```

6.1.2.21 geometric_sum

```
template<typename Integers , size_t deg> using aerobus::geometric_sum = typedef taylor<Integers, internal::geom_coeff, deg> \frac{1}{1-x} \text{ zero development of } \frac{1}{1-x}
```

Template Parameters

Integers	Ring type (for example i64)	
deg Generated by D	taylor approximation degree	

6.1.2.22 Inp1

```
template<typename Integers , size_t deg> using aerobus::lnp1 = typedef taylor<Integers, internal::lnp1_coeff, deg> \ln(1+x)
```

Template Parameters

T	Ring type (for example i64)
deg	taylor approximation degree

6.1.2.23 make_q32_t

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

helper type: make a fraction from numerator and denominator

Template Parameters

р	numerator
q	denominator

6.1.2.24 make_q64_t

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

helper type: make a fraction from numerator and denominator

Template Parameters

р	numerator
q	denominator

6.1.2.25 makefraction_t

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

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

Ring	the base ring
v1	value 1 in Ring
v2	value 2 in Ring

6.1.2.26 mulfractions_t

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

helper type: multiplies two fractions

Template Parameters

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

6.1.2.27 pi64

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

polynomial with 64 bits integers coefficients

6.1.2.28 PI fraction

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

6.1.2.29 pow_t

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

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

Template Parameters

T	(some ring type, such as aerobus::i64)
р	must be an instantiation of T::val
n	power

6.1.2.30 pq64

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

polynomial with 64 bits rationals coefficients

6.1.2.31 q32

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

32 bits rationals rationals with 32 bits numerator and denominator

6.1.2.32 q64

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

64 bits rationals rationals with 64 bits numerator and denominator

6.1.2.33 sin

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

Template Parameters

Integers	Ring type (for example i64)
deg	taylor approximation degree

6.1.2.34 sinh

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

Template Parameters

Integers	Ring type (for example i64)
deg	taylor approximation degree

6.1.2.35 SQRT2_fraction

```
2, 2, 2, 2, 2, 2, 2, 2, 2>
```

approximation of $\sqrt{2}$

6.1.2.36 SQRT3_fraction

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

approximation of

6.1.2.37 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 king (signed) – as types.

Template Parameters

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

6.1.2.38 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 king (unsigned) – as types.

Template Parameters

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

6.1.2.39 tan

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

tan(x) tangent

Integers	Ring type (for example i64)
deg	taylor approximation degree

6.1.2.40 tanh

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

tanh(x) hyperbolic tangent

Template Parameters

Integers	Ring type (for example i64)
deg	taylor approximation degree

6.1.2.41 taylor

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

Template Parameters

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

6.1.2.42 vadd_t

```
template<typename... vals>
using aerobus::vadd_t = typedef 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 Parameters



6.1.2.43 vmul_t

template<typename... vals>

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

multiplies multiplie values (v1 + v2 + ... + vn) vals must have same "enclosing_type" and "enclosing_type" must have an mul_t binary operator

Template Parameters

```
...vals
```

6.1.3 Function Documentation

6.1.3.1 aligned_malloc()

'portable' aligned allocation of count elements of type T

Template Parameters

```
T the type of elements to store
```

Parameters

count	the number of elements
alignment	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

```
T 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

FloatType	(double or float for example)
T	(aerobus::i64 for example)
n	

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

Т	(aerobus::i32 for example)
k	
n	

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

T	(aerobus::i64 for example)
i	

6.2 aerobus::internal Namespace Reference

internal implementations, subject to breaking changes without notice

Classes

```
    struct FractionField

\bullet \; \mathsf{struct} \, \_\mathsf{FractionField} < \mathbf{Ring}, \mathbf{std} :: \mathsf{enable} \_\mathsf{if} \_\mathsf{t} < \mathbf{Ring} :: \mathsf{is} \_\mathsf{euclidean} \_\mathsf{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 _{\bf is\_prime}< 5, i >

    struct _is_prime< 7, i >

• struct _is_prime< n, i, std::enable_if_t<(n !=2 &&n !=3 &&n % 2 !=0 &&n % 3==0)>>
• struct is prime < n, i, std::enable if t < (n != 2 \&\&n \% 2==0) > >
• struct _is_prime< n, i, std::enable_if_t<(n % i==0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&i *i > n)> >
• struct _is_prime< n, i, std::enable_if_t<(n %(i+2) !=0 &&n % i !=0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0
  &&(i *i<=n))> >
• struct is prime< n, i, std::enable if t<(n %(i+2)==0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&i *i<=n)>

    struct _is_prime< n, i, std::enable_if_t<(n >=9 &&i *i > n)> >

· struct alternate

    struct alternate < T, k, std::enable if t < k % 2 !=0 > >

    struct alternate< T, k, std::enable_if_t< k % 2==0 >>

· struct asin coeff

    struct asin coeff helper

struct asin_coeff_helper< T, i, std::enable_if_t<(i &1)==0 >>
struct asin_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >>

    struct asinh_coeff

· struct asinh_coeff_helper

    struct asinh_coeff_helper< T, i, std::enable_if_t<(i &1)==0 >>

• struct asinh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >
· struct atan coeff

    struct atan_coeff_helper

- struct atan_coeff_helper< T, i, std::enable_if_t<(i &1)==0 >>

    struct atan_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >>

· struct atanh coeff
· struct atanh coeff helper
struct atanh_coeff_helper< T, i, std::enable_if_t<(i &1)==0 >>
struct atanh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >>

    struct bell_helper

    struct bell_helper< T, 0 >

    struct bell helper< T, 1 >

struct bell_helper< T, n, std::enable_if_t<(n > 1)>>
· struct bernoulli

    struct bernoulli < T, 0 >

• struct bernoulli_coeff

    struct bernoulli helper

    struct bernoulli_helper< T, accum, m, m >

    struct bernstein helper
```

struct bernstein_helper< 0, 0 >

struct pop_front_h

```
• 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 >>

     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 Inp1_coeff

struct Inp1_coeff< T, 0 >

    struct make_taylor_impl

    struct make_taylor_impl< T, coeff_at, std::integer_sequence< size_t, ls... >>
```

```
· struct pow
• struct pow< T, p, n, std::enable_if_t< n==0 > >

    struct pow< T, p, n, std::enable_if_t<(n % 2==1)>>

    struct pow< T, p, n, std::enable_if_t<(n > 0 &&n % 2==0)> >

• struct pow scalar

    struct remove_h

    struct sh_coeff

· struct sh coeff helper
• struct sh coeff helper< T, i, std::enable if t<(i &1)==0>>

    struct sh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >>

    struct sin coeff

• struct sin_coeff_helper
struct sin_coeff_helper< T, i, std::enable_if_t<(i &1)==0 >>

    struct sin_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >>

    struct split h

    struct split h< 0, L1, L2 >

• struct stirling_helper

    struct stirling_helper< T, 0, 0 >

struct stirling_helper< T, 0, n, std::enable_if_t<(n > 0)>>
struct stirling_helper< T, n, 0, std::enable_if_t<(n > 0)> >
• struct stirling helper < T, n, k, std::enable if t<(k > 0) &&(n > 0)>

    struct tan coeff

• struct tan_coeff_helper
struct tan_coeff_helper< T, i, std::enable_if_t<(i % 2) !=0 >>

    struct tan_coeff_helper< T, i, std::enable_if_t<(i % 2)==0 >>

· struct tanh coeff
• struct tanh_coeff_helper
struct tanh_coeff_helper< T, i, std::enable_if_t<(i % 2) !=0 >>

    struct tanh_coeff_helper< T, i, std::enable_if_t<(i % 2)==0 >>

struct type_at

    struct type_at< 0, T, Ts... >

· struct vadd
struct vadd< v1 >
struct vadd< v1, vals... >
· struct vmul
struct vmul< v1 >
```

Typedefs

struct vmul< v1, vals... >

```
    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... ls>
 constexpr auto index_sequence_reverse (std::index_sequence< ls... > const &) -> decltype(std::index_
 sequence< sizeof...(ls) - 1U - ls... >{})

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()

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
```

deg	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	index of polynomial (between 0 and m)
n	n	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

dea	degree of polynomial
acg	acgree of polyflorinal
_	

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
```

```
deg 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

```
deg 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

```
deg 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
```

deg 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

deg degree of polynomial

6.3.3 Enumeration Type Documentation

6.3.3.1 hermite_kind

 $\verb"enum" aerobus::known_polynomials::hermite_kind"$

Enumerator



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_t<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

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'.

```
a0 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

```
\label{lem:continued} \begin{split} & template {<} int64\_t \ a0, \ int64\_t... \ rest{>} \\ & struct \ aerobus::ContinuedFraction {<} \ a0, \ rest... \ > \end{split}
```

specialization for multiple coefficients (strictly more than one)

Template Parameters

a0	integer (int64_t)
rest	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 It_t = typename It< 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>>

v1	a value in i <mark>32</mark>
v2	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 of represented values
```

```
    template < auto x >
```

```
using inject_constant_t = val< static_cast< int64_t >(x)>
```

• template<typename v >

```
using inject_ring_t = v
```

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

• using zero = val< 0 >

constant zero

• using one = val< 1 >

constant one

• template<typename v1 , typename v2 >

using add_t = typename add< v1, v2 >::type

template<typename v1 , typename v2 >

using sub_t = typename sub< v1, v2 >::type

template<typename v1 , typename v2 >

using mul t = typename mul < v1, v2 >::type

• template<typename v1 , typename v2 >

using div_t = typename div < v1, v2 >::type

```
template<typename v1, typename v2 > using mod_t = typename remainder< v1, v2 >::type
template<typename v1, typename v2 > using gt_t = typename gt< v1, v2 >::type
template<typename v1, typename v2 > using lt_t = typename lt< v1, v2 >::type
template<typename v1, typename v2 > using eq_t = typename eq< v1, v2 >::type
template<typename v1, typename v2 > using 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

```
v a value in i64
```

8.9.2.8 inner_type

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

type of represented values

8.9.2.9 It 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 It 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

```
n
```

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
```

substraction of two polynomials

```
• template<typename v1 , typename v2 >
  using mul_t = typename mul < v1, v2 >::type
     multiplication of two polynomials

    template<typename v1 , typename v2 >

  using eq_t = typename eq_helper< v1, v2 >::type
     equality operator
• template<typename v1 , typename v2 >
  using lt_t = typename lt_helper< v1, v2 >::type
     strict less operator
• template<typename v1 , typename v2 >
  using gt_t = typename gt_helper< v1, v2 >::type
     strict greater operator

    template<typename v1 , typename v2 >

  using div t = typename div < v1, v2 >::q type
     division operator

    template<typename v1 , typename v2 >

  using mod_t = typename div_helper< v1, v2, zero, v1 >::mod_type
     modulo operator
• template<typename coeff , size_t deg>
  using monomial t = typename monomial < coeff, deg >::type
     monomial : coeff X^{\wedge} 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

v1	
v2	

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



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

v1	
v2	

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

v1	
v2	

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

v1	
v2	

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

v1	
v2	

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

v1	
v2	

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

v1	
v2	

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
```

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

Template Parameters

coeff	
deg	

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

v1	
v2	

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



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



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
```

substraction of two polynomials

Template Parameters

v1	
v2	

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

```
v a value in polynomial::val
```

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 Z/2Z.

```
#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 >
• 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 > >
     substraction 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
  using inject_constant_t = val< typename Ring::template inject_constant_t < x > >

    template<typename v >

  using inject ring t = val< v >
```

Static Public Attributes

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

8.13.1 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 Z/2Z.

Template Parameters

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

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

V	1	a value in quotient ring
V2	2	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

v1	a value in quotient ring
v2	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> >
```

substraction operator

Template Parameters

v1	a value in quotient ring
v2	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

```
v1 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

v1	a value in quotient ring
v2	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

```
v1 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
```

8.14.2 Member Typedef Documentation

8.14.2.1 head

index

```
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

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



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

index	
T	

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

T

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



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

```
index
```

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

```
x 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()

cast x into valueRing

Template Parameters

```
valueRing double for example
```

8.17.3.2 get()

```
template<iint32_t x>
template<typename valueType >
static constexpr valueType aerobus::i32::val< x >::get ( ) [inline], [static], [constexpr]
```

cast x into valueType

Template Parameters

```
valueType | 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 inner_type = int32_t
        type of represented values
    using enclosing_type = i64
        enclosing ring type
    using is_zero_t = std::bool_constant< x==0 >
        is value zero
```

Static Public Member Functions

```
    template<typename valueType > static constexpr 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

```
x an actual integer
```

8.18.2 Member Typedef Documentation

8.18.2.1 enclosing_type

```
template<iint64_t x>
using aerobus::i64::val< x >::enclosing_type = i64
enclosing ring type
```

8.18.2.2 inner_type

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

type of represented values

8.18.2.3 is_zero_t

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

is value zero

8.18.3 Member Function Documentation

8.18.3.1 eval()

cast value in valueRing

Template Parameters

```
valueRing (double for example)
```

8.18.3.2 get()

```
\texttt{template} {<} \texttt{int64\_t} \texttt{ x} {>}
```

cast value in valueType

Template Parameters

```
valueType (double for example)
```

8.18.3.3 to_string()

string representation

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

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

coeffN	high degree coefficient
coeffs	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_\to at<index>::type
```

type of coefficient at index

Template Parameters

```
index
```

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()

evaluates polynomial seen as a function operating on ValueRing

Template Parameters

valueRing usually float or double

Parameters

```
x 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 + ... + 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 \leftarrow
::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::val< x > Struct Template Reference

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

Public Types

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

Static Public Member Functions

- template < typename value Type > static constexpr value Type 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::val< x >::enclosing_type = zpz
enclosing ring type
```

8.21.1.2 is zero t

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

8.21.2 Member Function Documentation

8.21.2.1 eval()

8.21.2.2 get()

```
template<int32_t p>
template<int32_t x>
template<typename valueType >
static constexpr valueType aerobus::zpz::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::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::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

```
coeffN
```

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_\top 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()

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 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
     substraction 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 It_t = typename It < 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::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
```

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<iint32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz::add_t = typename add<v1, v2>::type
```

addition operator

Template Parameters

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

8.23.2.2 div_t

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

division operator

Template Parameters

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

8.23.2.3 eq_t

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

equality operator (type)

Template Parameters

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

8.23.2.4 gcd_t

```
template<iint32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz::gcd_t = gcd_t<i32, v1, v2>
```

greatest common divisor

Template Parameters

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

8.23.2.5 gt_t

```
template<int32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz::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::inject_constant_t = val<static_cast<int32_t>(x)>
```

8.23.2.7 inner_type

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

8.23.2.8 lt_t

```
template<iint32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz::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::mod_t = typename remainder<v1, v2>::type
```

modulo operator

Template Parameters

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

8.23.2.10 mul_t

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

multiplication operator

Template Parameters

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

8.23.2.11 one

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

8.23.2.12 pos_t

```
template<iint32_t p>
template<typename v1 >
using aerobus::zpz::pos_t = typename pos<v1>::type
```

positivity operator (type)

Template Parameters

```
v1 a value in zpz::val
```

8.23.2.13 sub t

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

substraction operator

Template Parameters

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

8.23.2.14 zero

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

8.23.3.4 is_field

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

8.23.3.5 lt_v

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

strictly smaller operator (booleanvalue)

Template Parameters

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

8.23.3.6 pos_v

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

positivity operator (boolean value)

Template Parameters

```
v1 a value in zpz::val
```

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

• src/aerobus.h

Chapter 9

File Documentation

9.1 README.md File Reference

9.2 src/aerobus.h File Reference

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

9.3 aerobus.h

Go to the documentation of this file.

```
00001 // -*- lsst-c++ -*-
00002 #ifndef __INC_AEROBUS__ // NOLINT
00003 #define __INC_AEROBUS__
00004
00005 #include <cstdint>
00006 #include <cstddef>
00007 #include <cstring>
00008 #include <type_traits>
00009 #include <utility>
00010 #include <algorithm>
00011 #include <functional>
00012 #include <string>
00013 #include <concepts> // NOLINT
00014 #include <array>
00015
00018 #ifdef _MSC_VER
00019 #define ALIGNED(x) __declspec(align(x))
00020 #define INLINED __forceinline
00021 #else
00022 #define ALIGNED(x) __attribute__((aligned(x)))
00023 #define INLINED __attribute__((always_inline)) inline
00024 #endif
00025
00027
00029
```

92 File Documentation

```
00031
00032 // aligned allocation
00033 namespace aerobus {
00040
          template<typename T>
00041
           T* aligned_malloc(size_t count, size_t alignment) {
               #ifdef _MSC_VER
return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
00042
00043
00044
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>
           concept IsRing = requires {
00054
00055
               typename R::one;
               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>
           concept IsEuclideanDomain = IsRing<R> && requires {
00064
00065
               typename R::template div_t<typename R::one, typename R::one>;
00066
               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>;
00067
00068
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;
08000
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
               template<template<ttypename...> typename TT, typename... Ts>
struct is_instantiation_of<TT, TT<Ts...» : std::true_type { };</pre>
00089
00090
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");</pre>
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
               template <size_t i, typename... Ts>
using type_at_t = typename type_at<i, Ts...>::type;
00105
00106
00107
00108
00109
               template<size_t n, size_t i, typename E = void>
00110
               struct _is_prime {};
00111
00112
               template<size t i>
               struct _{is\_prime<0, i> \{}
00113
                   static constexpr bool value = false;
00114
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
```

9.3 aerobus.h

```
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
              template<size t i>
00137
              struct _is_prime<7, i> {
00138
00139
                  static constexpr bool value = true;
00140
00141
00142
              {\tt template}{<} {\tt size\_t n, size\_t i}{\gt}
              struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)» {
    static constexpr bool value = false;</pre>
00143
00144
00146
              template<size_t n, size_t i>
00147
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>
              struct _is_prime<n, i, std::enable_if_t<(
    n % i == 0 &&</pre>
00158
00159
00160
                  n >= 9 &&
00161
                  n % 3 != 0 &&
                  n % 2 != 0 &&
00162
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 &&
n >= 9 &&
00181
00182
                       n % 3 != 0 &&
                       n % 2 != 0 &&
00184
                       (i * i \le n)) \gg {
00185
                   static constexpr bool value = _is_prime<n, i+6>::value;
00186
              };
00187
00188
          } // namespace internal
00189
00192
          template<size t n>
00193
          struct is_prime {
00195
              static constexpr bool value = internal::_is_prime<n, 5>::value;
00196
00197
00201
          template<size t n>
00202
          static constexpr bool is_prime_v = is_prime<n>::value;
00203
00204
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
00219
              template<typename Ring, typename E = void>
00220
              struct qcd;
00221
00222
              template<typename Ring>
              struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
00223
00224
                  template<typename A, typename B, typename E = void>
```

94 File Documentation

```
struct gcd_helper {};
00226
00227
                  // B = 0, A > 0
00228
                  template<typename A, typename B>
                  struct gcd_helper<A, B, std::enable_if_t<
    ((B::is_zero_t::value) &&</pre>
00229
00230
00231
                          (Ring::template gt_t<A, typename Ring::zero>::value))» {
00232
                       using type = A;
00233
                  } ;
00234
                  // B = 0. A < 0
00235
                  template<typename A, typename B>
struct gcd_helper<A, B, std::enable_if_t<</pre>
00236
00237
                      ((B::is_zero_t::value) &&
00238
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
                       » {
                  private: // NOLINT
00248
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>
              struct vmul<v1, vals...> {
00269
00270
                 using type = typename v1::enclosing_type::template mul_t<v1, typename
     vmul<vals...>::type>;
00271
             } ;
00272
00273
              template<tvpename 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> {
                using type = v1;
00288
00289
              };
00290
          } // namespace internal
00291
00294
          template<typename T, typename A, typename B>
00295
          using gcd_t = typename internal::gcd<T>::template type<A, B>;
00296
00300
          template<typename... vals>
00301
          using vadd_t = typename internal::vadd<vals...>::type;
00302
00306
          template<typename... vals>
00307
          using vmul_t = typename internal::vmul<vals...>::type;
00308
00312
          template < typename val>
00313
          requires IsEuclideanDomain<typename val::enclosing type>
00314
          using abs_t = std::conditional_t<
00315
                           val::enclosing_type::template pos_v<val>,
                           val, typename val::enclosing_type::template sub_t<typename</pre>
     val::enclosing_type::zero, val»;
00317 } // namespace aerobus
00318
00319 namespace aerobus {
```

9.3 aerobus.h

```
template<typename Ring, typename X>
          requires IsRing<Ring>
00325
00326
          struct Quotient {
00329
              template <typename V>
00330
              struct val {
              public:
00331
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<tvpename 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>
              using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00369
00370
00374
              template<typename v1, typename v2>
              static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00375
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
              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 {
         template <typename... Ts>
00413
00414
          struct type_list;
00415
00416
          namespace internal {
00417
              template <typename T, typename... Us>
00418
              struct pop_front_h {
                  using tail = type_list<Us...>;
using head = T;
00419
00420
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");</pre>
                  using a = typename L2::pop_front::type;
00427
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;
                  using tail = typename split_h<index - 1, c, b>::tail;
00433
00434
00435
              template <typename L1, typename L2>
struct split_h<0, L1, L2> {
    using head = L1;
00436
00437
00438
                  using tail = L2;
00439
00440
00441
00442
              template <size_t index, typename L, typename T>
              struct insert_h {
00443
                  static_assert(index <= L::length, "index ouf of bounds");
00444
00445
                  using s = typename L::template split<index>;
00446
                  using left = typename s::head;
00447
                  using right = typename s::tail;
00448
                  using 11 = typename left::template push_back<T>;
00449
                  using type = typename ll::template concat<right>;
00450
              };
```

96 File Documentation

```
00451
00452
               template <size_t index, typename L>
00453
               struct remove_h {
00454
                  using s = typename L::template split<index>;
                   using left = typename s::head;
00455
                   using right = typename s::tail;
00456
                   using rr = typename right::pop_front::tail;
00458
                   using type = typename left::template concat<rr>;
00459
          } // namespace internal
00460
00461
00465
          template <typename... Ts>
00466
          struct type_list {
00467
           private:
00468
              template <typename T>
00469
               struct concat_h;
00470
00471
               template <typename... Us>
               struct concat_h<type_list<Us...» {
00472
00473
                  using type = type_list<Ts..., Us...>;
00474
00475
           public:
00476
00478
              static constexpr size_t length = sizeof...(Ts);
00479
00482
               template <typename T>
00483
               using push_front = type_list<T, Ts...>;
00484
00487
               template <size_t index>
00488
               using at = internal::type_at_t<index, Ts...>;
00489
00491
               struct pop front {
00493
                  using type = typename internal::pop_front_h<Ts...>::head;
00495
                   using tail = typename internal::pop_front_h<Ts...>::tail;
00496
00497
              template <typename T>
using push_back = type_list<Ts..., T>;
00500
00502
00505
               template <typename U>
00506
               using concat = typename concat_h<U>::type;
00507
00510
               template <size t index>
00511
               struct split {
               private:
00512
00513
                   using inner = internal::split_h<index, type_list<>, type_list<Ts...»;</pre>
00514
00515
                public:
                   using head = typename inner::head;
using tail = typename inner::tail;
00516
00517
00518
               };
00519
00523
               template <typename T, size_t index>
00524
               using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00525
00528
               template <size_t index>
              using remove = typename internal::remove_h<index, type_list<Ts...»::type;
00529
00530
          };
00531
00533
          template <>
          struct type_list<> {
00534
              static constexpr size_t length = 0;
00535
00536
00537
               template <typename T>
00538
               using push_front = type_list<T>;
00539
00540
               template <typename T>
00541
               using push_back = type_list<T>;
00542
00543
               template <typename U>
00544
               using concat = U;
00545
00546
               // TODO(jewave): assert index == 0
              template <typename T, size_t index>
using insert = type_list<T>;
00547
00548
00549
          };
00550 } // namespace aerobus
00551
00552 // i32
00553 namespace aerobus {
         struct i32 {
00555
              using inner_type = int32_t;
00559
               template<int32_t x>
00560
               struct val {
                  using enclosing_type = i32;
static constexpr int32_t v = x;
00562
00564
00565
```

```
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>
                  static constexpr valueRing eval(const valueRing& v) {
00582
00583
                      return static_cast<valueRing>(x);
00584
00585
              };
00586
              using zero = val<0>;
using one = val<1>;
00588
00590
00592
              static constexpr bool is field = false;
              static constexpr bool is_euclidean_domain = true;
00594
00598
              template<auto x>
              using inject_constant_t = val<static_cast<int32_t>(x)>;
00599
00600
00601
              {\tt template}{<}{\tt typename}\ {\tt 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
00612
                  using type = val<v1::v - v2::v>;
00613
00614
              template<typename v1, typename v2>
00615
              struct mul {
00616
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 {
                  using type = val<v1::v % v2::v>;
00627
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
              template<typename v1, typename v2>
00636
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 {
                  using type = std::bool_constant<(v1::v > 0)>;
00647
00648
              };
00649
00650
           public:
00656
              template<typename v1, typename v2>
00657
              using add_t = typename add<v1, v2>::type;
00658
              template<typename v1, typename v2>
00664
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
```

```
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
              template<typename v>
00742
             static constexpr bool pos_v = pos_t<v>::value;
00743
00744 } // namespace aerobus
00745
00746 // i64
00747 namespace aerobus {
00749
        struct i64 {
00751
              using inner_type = int64_t;
00754
              template<int64_t x>
00755
              struct val {
                using inner_type = int32_t;
00757
00759
                  using enclosing_type = i64;
00761
                 static constexpr int64_t v = x;
00762
00765
                  template<typename valueType>
00766
                  static constexpr valueType get() {
00767
                      return static_cast<valueType>(x);
00768
00769
00771
                  using is_zero_t = std::bool_constant<x == 0>;
00772
00774
                  static std::string to_string() {
00775
                      return std::to_string(x);
00776
00777
00780
                  template<typename valueRing>
00781
                  static constexpr valueRing eval(const valueRing& v) {
00782
                     return static_cast<valueRing>(x);
00783
                  }
00784
              };
00785
00789
              template<auto x>
00790
              using inject_constant_t = val<static_cast<int64_t>(x)>;
00791
00796
              template<typename v>
00797
              using inject_ring_t = v;
00798
00800
              using zero = val<0>;
00802
              using one = val<1>;
00804
              static constexpr bool is_field = false;
00806
              static constexpr bool is_euclidean_domain = true;
00807
00808
           private:
00809
              template<typename v1, typename v2>
00810
              struct add {
00811
                  using type = val<v1::v + v2::v>;
00812
00813
00814
              template<typename v1, typename v2>
00815
              struct sub {
                 using type = val<v1::v - v2::v>;
00816
00817
00818
00819
              template<typename v1, typename v2>
00820
              struct mul {
                 using type = val<v1::v* v2::v>;
00821
00822
00823
00824
              template<typename v1, typename v2>
              struct div {
00825
                 using type = val<v1::v / v2::v>;
00826
00827
00828
00829
              template<typename v1, typename v2>
00830
              struct remainder {
00831
                  using type = val<v1::v% v2::v>;
00832
00833
```

```
00834
              template<typename v1, typename v2>
00835
00836
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00837
00838
00839
              template<tvpename v1, tvpename v2>
              struct lt {
00840
00841
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00842
00843
00844
              template<typename v1, typename v2>
00845
              struct eq {
00846
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00847
00848
00849
              template<typename v>
00850
              struct pos {
00851
                  using type = std::bool_constant<(v::v > 0)>;
00852
00853
00854
           public:
00859
              template<typename v1, typename v2>
00860
              using add_t = typename add<v1, v2>::type;
00861
00866
              template<typename v1, typename v2>
              using sub_t = typename sub<v1, v2>::type;
00867
00868
00873
              template<typename v1, typename v2>
00874
              using mul_t = typename mul<v1, v2>::type;
00875
00881
              template<typename v1, typename v2>
00882
              using div_t = typename div<v1, v2>::type;
00883
00888
              template<typename v1, typename v2>
00889
              using mod_t = typename remainder<v1, v2>::type;
00890
00896
              template<typename v1, typename v2>
              using gt_t = typename gt<v1, v2>::type;
00898
00903
              template<typename v1, typename v2>
00904
              static constexpr bool gt_v = gt_t<v1, v2>::value;
00905
              template<typename v1, typename v2>
using lt_t = typename lt<v1, v2>::type;
00911
00912
00913
              template<typename v1, typename v2>
00919
00920
              static constexpr bool lt_v = lt_t<v1, v2>::value;
00921
00927
              template<typename v1, typename v2>
00928
              using eq_t = typename eq<v1, v2>::type;
00929
00935
              template<typename v1, typename v2>
00936
              static constexpr bool eq_v = eq_t<v1, v2>::value;
00937
00943
              template<typename v1, typename v2>
00944
              using gcd_t = gcd_t < i64, v1, v2>;
00945
00950
              template<typename v>
00951
              using pos_t = typename pos<v>::type;
00952
00957
              template<typename v>
00958
              static constexpr bool pos_v = pos_t<v>::value;
00959
          };
00960 } // namespace aerobus
00961
00962 // z/pz
00963 namespace aerobus {
          template<int32_t p>
00968
00969
          struct zpz {
              using inner_type = int32_t;
00970
00971
              template<int32_t x>
00972
              struct val {
                  using enclosing_type = zpz;
static constexpr int32_t v = x % p;
00974
00976
00977
00978
                  template<typename valueType>
00979
                  static constexpr valueType get() { return static_cast<valueType>(x % p); }
00980
00981
                  using is_zero_t = std::bool_constant<x% p == 0>;
                  static std::string to_string() {
00982
00983
                      return std::to_string(x % p);
00984
00985
00986
                  template<typename valueRing>
00987
                  static constexpr valueRing eval(const valueRing& v) {
00988
                       return static_cast<valueRing>(x % p);
00989
                   }
```

```
00990
              };
00991
00992
              template<auto x>
00993
              using inject_constant_t = val<static_cast<int32_t>(x)>;
00994
00995
              using zero = val<0>:
              using one = val<1>;
00996
00997
              static constexpr bool is_field = is_prime::value;
00998
              static constexpr bool is_euclidean_domain = true;
00999
01000
           private:
              template<typename v1, typename v2>
01001
01002
              struct add {
                  using type = val<(v1::v + v2::v) % p>;
01003
01004
01005
              template<typename v1, typename v2>
01006
01007
              struct sub {
01008
                 using type = val<(v1::v - v2::v) % p>;
01009
01010
01011
              template<typename v1, typename v2>
01012
              struct mul {
                  using type = val<(v1::v* v2::v) % p>;
01013
01014
01015
01016
              template<typename v1, typename v2>
01017
              struct div {
                 using type = val<(v1::v% p) / (v2::v % p)>;
01018
01019
01020
01021
              template<typename v1, typename v2>
01022
              struct remainder {
01023
                  using type = val<(v1::v% v2::v) % p>;
01024
01025
01026
              template<typename v1, typename v2>
              struct qt {
01028
                 using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01029
01030
01031
              template<typename v1, typename v2>
01032
              struct lt {
01033
                  using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01034
01035
01036
              template<typename v1, typename v2>
              struct eq {
01037
                 using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01038
01039
01040
01041
              template<typename v1>
01042
              struct pos {
01043
                 using type = std::bool_constant<(v1::v > 0)>;
01044
              };
01045
01046
           public:
01050
              template<typename v1, typename v2>
01051
              using add_t = typename add<v1, v2>::type;
01052
01056
              template<typename v1, typename v2>
01057
              using sub_t = typename sub<v1, v2>::type;
01058
01062
              template<typename v1, typename v2>
01063
              using mul_t = typename mul<v1, v2>::type;
01064
01068
              template<typename v1, typename v2>
01069
              using div_t = typename div<v1, v2>::type;
01070
01074
              template<typename v1, typename v2>
01075
              using mod_t = typename remainder<v1, v2>::type;
01076
01080
              template<typename v1, typename v2>
01081
              using gt_t = typename gt<v1, v2>::type;
01082
01086
              template<typename v1, typename v2>
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01087
01088
01092
              template<typename v1, typename v2>
              using lt_t = typename lt<v1, v2>::type;
01093
01094
01098
              template<typename v1, typename v2>
01099
              static constexpr bool lt_v = lt_t<v1, v2>::value;
01100
01104
              template<typename v1, typename v2> ^{\circ}
01105
              using eq_t = typename eq<v1, v2>::type;
01106
```

```
template<typename v1, typename v2>
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01111
01112
01116
              template<typename v1, typename v2> ^{\circ}
              using gcd_t = gcd_t < i32, v1, v2>;
01117
01118
01121
              template<typename v1>
01122
              using pos_t = typename pos<v1>::type;
01123
01126
              template < typename v >
              static constexpr bool pos_v = pos_t<v>::value;
01127
01128
          };
01129 } // namespace aerobus
01130
01131 // polynomial
01132 namespace aerobus {
          // coeffN x^N + ..
01133
          template<typename Ring>
01138
01139
          requires IsEuclideanDomain<Ring>
01140
          struct polynomial {
01141
              static constexpr bool is_field = false;
01142
              static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01143
01147
              template<typename coeffN, typename... coeffs>
01148
              struct val {
                 using enclosing_type = polynomial<Ring>;
01150
01152
                   static constexpr size_t degree = sizeof...(coeffs);
01154
                  using aN = coeffN;
01156
                  using strip = val<coeffs...>;
                  using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01158
01160
                  static constexpr bool is zero v = is zero t::value;
01161
01162
01163
                  template<size_t index, typename E = void>
01164
                  struct coeff_at {};
01165
01166
                  template<size t index>
01167
                  struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))» {</pre>
                      using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01168
01169
01170
01171
                  template<size t index>
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))» {
01172
01173
                      using type = typename Ring::zero;
01174
                  };
01175
01176
               public:
01179
                  template<size_t index>
                  using coeff_at_t = typename coeff_at<index>::type;
01180
01181
01184
                  static std::string to_string() {
01185
                      return string_helper<coeffN, coeffs...>::func();
01186
01187
                  template<typename valueRing>
01192
                  static constexpr valueRing eval(const valueRing& x) {
   return horner_evaluation<valueRing, val>
01193
01194
01195
                               ::template inner<0, degree + 1>
01196
                               ::func(static_cast<valueRing>(0), x);
01197
                  }
01198
              }:
01199
01202
              template<typename coeffN>
01203
              struct val<coeffN> {
01205
                  using enclosing_type = polynomial<Ring>;
01207
                  static constexpr size_t degree = 0;
01208
                  using aN = coeffN;
01209
                  using strip = val<coeffN>;
01210
                  using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01211
01212
                  static constexpr bool is_zero_v = is_zero_t::value;
01213
01214
                  template<size_t index, typename E = void>
01215
                  struct coeff at {}:
01216
01217
                   template<size_t index>
01218
                  struct coeff_at<index, std::enable_if_t<(index == 0)» {</pre>
01219
                      using type = aN;
01220
                  };
01221
01222
                  template<size t index>
01223
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01224
                      using type = typename Ring::zero;
01225
                  };
01226
01227
                  template<size_t index>
01228
                  using coeff_at_t = typename coeff_at<index>::type;
```

```
01229
01230
                  static std::string to_string() {
01231
                       return string_helper<coeffN>::func();
01232
01233
01234
                  template<tvpename valueRing>
                  static constexpr valueRing eval(const valueRing& x) {
01235
01236
                       return static_cast<valueRing>(aN::template get<valueRing>());
01237
01238
              };
01239
01241
              using zero = val<typename Ring::zero>;
              using one = val<typename Ring::one>;
01243
01245
              using X = val<typename Ring::one, typename Ring::zero>;
01246
01247
              template<typename P, typename E = void>
01248
01249
              struct simplify;
01250
01251
              template <typename P1, typename P2, typename I>
01252
              struct add_low;
01253
01254
              template<typename P1, typename P2>
01255
              struct add {
01256
                  using type = typename simplify<typename add_low<
01257
01258
                  P2,
01259
                  internal::make_index_sequence_reverse<</pre>
01260
                  std::max(P1::degree, P2::degree) + 1
01261
                  »::type>::type;
01262
              };
01263
01264
              template <typename P1, typename P2, typename I>
01265
              struct sub_low;
01266
              template <typename P1, typename P2, typename I>
01267
01268
              struct mul low;
01269
01270
              template<typename v1, typename v2>
01271
              struct mul {
01272
                       using type = typename mul_low<
                          v1,
01273
01274
                           v2.
01275
                           internal::make_index_sequence_reverse<</pre>
01276
                           v1::degree + v2::degree + 1
01277
                           »::type;
01278
01279
01280
              template<typename coeff, size t deg>
01281
              struct monomial:
01283
              template<typename v, typename E = void>
01284
              struct derive_helper {};
01285
01286
              template<typename v>
              struct derive_helper<v, std::enable_if_t<v::degree == 0» {</pre>
01287
01288
                  using type = zero;
01289
01290
01291
              template<typename v>
01292
              struct derive helper<v, std::enable if t<v::degree != 0» {
01293
                  using type = typename add<
01294
                       typename derive_helper<typename simplify<typename v::strip>::type>::type,
01295
                       typename monomial<
01296
                           typename Ring::template mul_t<</pre>
01297
                               typename v::aN,
01298
                               typename Ring::template inject_constant_t<(v::degree)>
01299
01300
                           v::degree - 1
01301
                       >::type
01302
                  >::type;
01303
              } ;
01304
01305
              template<typename v1, typename v2, typename E = void>
01306
              struct eq helper {};
01307
01308
              template<typename v1, typename v2>
01309
              struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
01310
                  using type = std::false_type;
01311
01312
01313
01314
              template<typename v1, typename v2>
              struct eq_helper<v1, v2, std::enable_if_t<
01315
                  v1::degree == v2::degree &&
(v1::degree != 0 || v2::degree != 0) &&
01316
01317
01318
                  std::is same<
```

```
typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
                   std::false_type
01320
01321
                   >::value
01322
              >
01323
              > {
01324
                   using type = std::false type;
01325
               };
01326
01327
               template<typename v1, typename v2>
               struct eq_helper<v1, v2, std::enable_if_t<
    v1::degree == v2::degree &&</pre>
01328
01329
                   (v1::degree != 0 || v2::degree != 0) &&
01330
01331
                   std::is_same<
01332
                   typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
                   std::true_type
01333
01334
01335
               » {
01336
                   using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01337
01338
               template<typename v1, typename v2>
struct eq_helper<v1, v2, std::enable_if_t<
    v1::degree == v2::degree &&</pre>
01339
01340
01341
                   (v1::degree == 0)
01342
01343
               » {
01344
                   using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01345
               };
01346
01347
               template<typename v1, typename v2, typename E = void>
01348
               struct lt_helper {};
01349
01350
               template<typename v1, typename v2>
01351
               struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
01352
                   using type = std::true_type;
01353
01354
              template<typename v1, typename v2>
struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01355
01356
01357
                   using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01358
01359
              template<typename v1, typename v2>
struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
01360
01361
                   using type = std::false_type;
01362
01363
01364
01365
               template<typename v1, typename v2, typename E = void>
01366
               struct gt_helper {};
01367
01368
               template<typename v1, typename v2> \,
               struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
01369
01370
                   using type = std::true_type;
01371
01372
               01373
01374
01375
                  using type = std::false_type;
01376
01377
01378
               template<typename v1, typename v2> \,
01379
               \label{lem:struct_gt_helper_v1, v2, std::enable_if_t<(v1::degree < v2::degree) } \\
01380
                  using type = std::false_type;
01381
01382
01383
               // when high power is zero : strip
01384
               template<typename P>
01385
               struct simplify<P, std::enable_if_t<
01386
                   std::is_same<
01387
                   typename Ring::zero,
01388
                   typename P::aN
01389
                   >::value && (P::degree > 0)
01390
01391
                   using type = typename simplify<typename P::strip>::type;
01392
               };
01393
01394
               // otherwise : do nothing
01395
               template<typename P>
01396
               struct simplify<P, std::enable_if_t<</pre>
01397
                   !std::is_same<
01398
                   typename Ring::zero.
01399
                   typename P::aN
01400
                   >::value && (P::degree > 0)
01401
01402
                   using type = P;
01403
               };
01404
01405
               // do not simplify constants
```

```
template<typename P>
                                 struct simplify<P, std::enable_if_t<P::degree == 0» {</pre>
01407
01408
                                          using type = P;
01409
01410
                                 // addition at
01411
01412
                                 template<typename P1, typename P2, size_t index>
01413
                                 struct add_at {
                                        using type =
01414
01415
                                                    typename Ring::template add_t<</pre>
                                                             typename P1::template coeff_at_t<index>,
01416
01417
                                                             typename P2::template coeff_at_t<index>>;
01418
                                 };
01419
01420
                                 template<typename P1, typename P2, size_t index>
01421
                                 using add_at_t = typename add_at<P1, P2, index>::type;
01422
01423
                                 template<typename P1, typename P2, std::size_t... I>
                                 struct add_low<P1, P2, std::index_sequence<I...» {
01424
                                        using type = val<add_at_t<P1, P2, I>...>;
01425
01426
01427
                                // substraction at
01428
                                 template<typename P1, typename P2, size_t index>
01429
01430
                                 struct sub_at {
01431
                                        using type =
01432
                                                    typename Ring::template sub_t<
01433
                                                             typename P1::template coeff_at_t<index>,
01434
                                                             typename P2::template coeff_at_t<index>>;
01435
                                 };
01436
01437
                                 template<typename P1, typename P2, size_t index>
01438
                                 using sub_at_t = typename sub_at<P1, P2, index>::type;
01439
                                 template<typename P1, typename P2, std::size_t... I>
struct sub_low<P1, P2, std::index_sequence<I...» {
    using type = val<sub_at_t<P1, P2, I>...>;
01440
01441
01442
01443
01444
01445
                                 template<typename P1, typename P2>
01446
                                 struct sub {
01447
                                         using type = typename simplify<typename sub low<
                                          P1,
01448
01449
01450
                                         internal::make_index_sequence_reverse<
01451
                                          std::max(P1::degree, P2::degree) + 1
01452
                                          »::type>::type;
01453
                                };
01454
                                 // multiplication at
01455
01456
                                 template<typename v1, typename v2, size_t k, size_t index, size_t stop>
01457
                                 struct mul_at_loop_helper {
01458
                                          using type = typename Ring::template add_t<
                                                   typename Ring::template mul_t<
typename v1::template coeff_at_t<index>,
01459
01460
01461
                                                   typename v2::template coeff_at_t<k - index>
01462
01463
                                                    typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
01464
01465
                                };
01466
                                 template<typename v1, typename v2, size_t k, size_t stop>
struct mul_at_loop_helper<v1, v2, k, stop, stop> {
01467
01468
                                         using type = typename Ring::template mul_t<
01469
01470
                                                    typename v1::template coeff_at_t<stop>,
01471
                                                   typename v2::template coeff_at_t<0>>;
01472
                                };
01473
01474
                                 template <typename v1, typename v2, size_t k, typename E = void>
01475
                                 struct mul_at {};
01476
01477
                                 template<typename v1, typename v2, size_t k>
                                  \texttt{struct mul\_at} < \texttt{v1, v2, k, std} :: \texttt{enable\_if\_t} < (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{v1} :: \texttt{degree} + \texttt{v2} :: \texttt{degree}) \\ \texttt{ } \times (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{v1} :: \texttt{degree}) \\ \texttt{ } \times (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{v1} :: \texttt{degree}) \\ \texttt{ } \times (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{v1} :: \texttt{degree}) \\ \texttt{ } \times (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{v1} :: \texttt{degree}) \\ \texttt{ } \times (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{v1} :: \texttt{degree}) \\ \texttt{ } \times (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{v1} :: \texttt{degree}) \\ \texttt{ } \times (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{v2} :: \texttt{degree}) \\ \texttt{ } \times (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{v1} :: \texttt{degree}) \\ \texttt{ } \times (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{v2} :: \texttt{degree}) \\ \texttt{ } \times (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{v2} :: \texttt{degree}) \\ \texttt{ } \times (\texttt{k} < \texttt{0}) \ \mid \mid \ (\texttt{k} > \texttt{0}) \ \mid \ (\texttt
01478
01479
                                         using type = typename Ring::zero;
01480
01481
01482
                                 template<typename v1, typename v2, size_t k>
01483
                                 01484
                                          using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01485
01486
01487
                                 template<typename P1, typename P2, size_t index>
01488
                                 using mul_at_t = typename mul_at<P1, P2, index>::type;
01489
01490
                                 template<typename P1, typename P2, std::size_t... I>
                                 struct mul_low<P1, P2, std::index_sequence<I...» {
    using type = val<mul_at_t<P1, P2, I>...>;
01491
01492
```

```
01493
              };
01494
01495
              // division helper
01496
              template< typename A, typename B, typename Q, typename R, typename E = void>
01497
              struct div_helper {};
01498
01499
              template<typename A, typename B, typename Q, typename R>
01500
              struct div_helper<A, B, Q, R, std::enable_if_t
01501
                 (R::degree < B::degree) ||
01502
                   (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
                  using q_type = Q;
01503
01504
                  using mod_type = R;
01505
                  using gcd type = B;
01506
01507
01508
              template<typename A, typename B, typename Q, typename R>
              01509
01510
                   !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
01511
01512
               private: // NOLINT
01513
                  using rN = typename R::aN;
01514
                  using bN = typename B::aN;
01515
                  using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
     B::degree>::type;
01516
                  using rr = typename sub<R, typename mul<pT, B>::type>::type;
                  using qq = typename add<Q, pT>::type;
01517
01518
               public:
01519
01520
                  using q_type = typename div_helper<A, B, qq, rr>::q_type;
01521
                  using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
01522
                  using gcd_type = rr;
01523
              }:
01524
01525
              template<typename A, typename B>
01526
              struct div {
                  static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
01527
                  using q_type = typename div_helper<A, B, zero, A>::q_type; using m_type = typename div_helper<A, B, zero, A>::mod_type;
01528
01529
01530
              };
01531
01532
              template<typename P>
01533
              struct make_unit {
                  using type = typename div<P, val<typename P::aN>>::q_type;
01534
01535
01536
01537
              template<typename coeff, size_t deg>
01538
              struct monomial {
01539
                  using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
01540
01541
01542
              template<typename coeff>
01543
              struct monomial < coeff, 0 > {
01544
                  using type = val<coeff>;
01545
01546
01547
              template<typename valueRing, typename P>
01548
              struct horner_evaluation {
01549
                  template<size_t index, size_t stop>
01550
                  struct inner {
01551
                      static constexpr valueRing func(const valueRing& accum, const valueRing& x) {
01552
                          constexpr valueRing coeff =
                               static cast<valueRing>(P::template coeff at t<P::degree - index>::template
01553
     get<valueRing>());
01554
                           return horner_evaluation<valueRing, P>::template inner<index + 1, stop>::func(x *
     accum + coeff, x);
01555
01556
                  };
01557
01558
                  template<size t stop>
                  struct inner<stop, stop> {
01560
                     static constexpr valueRing func(const valueRing& accum, const valueRing& x) {
01561
                           return accum;
01562
                  };
01563
01564
              };
01565
01566
              template<typename coeff, typename... coeffs>
01567
              struct string_helper {
01568
                  static std::string func() {
                      std::string tail = string_helper<coeffs...>::func();
std::string result = "";
01569
01570
01571
                       if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
01572
                           return tail;
01573
                       } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
                          if (sizeof...(coeffs) == 1) {
   result += "x";
01574
01575
                           } else {
01576
```

```
result += "x^" + std::to_string(sizeof...(coeffs));
01578
01579
                       } else {
01580
                          if (sizeof...(coeffs) == 1) {
                               result += coeff::to_string() + " x";
01581
01582
                           } else {
01583
                              result += coeff::to_string()
01584
                                       + " x^" + std::to_string(sizeof...(coeffs));
01585
01586
                      }
01587
                      if (!tail.empty()) {
    result += " + " + tail;
01588
01589
01590
01591
01592
                      return result;
01593
                  }
01594
              };
01595
01596
              template<typename coeff>
01597
              struct string_helper<coeff> {
01598
                  static std::string func() {
                      if (!std::is_same<coeff, typename Ring::zero>::value) {
01599
                          return coeff::to_string();
01600
01601
                      } else {
                         return "";
01602
01603
01604
01605
              };
01606
01607
           public:
01610
              template<typename P>
01611
              using simplify_t = typename simplify<P>::type;
01612
01616
              template<typename v1, typename v2>
01617
              using add_t = typename add<v1, v2>::type;
01618
01622
              template<typename v1, typename v2>
01623
              using sub_t = typename sub<v1, v2>::type;
01624
01628
              template<typename v1, typename v2> \,
01629
              using mul_t = typename mul<v1, v2>::type;
01630
01634
              template<typename v1, typename v2>
01635
              using eq_t = typename eq_helper<v1, v2>::type;
01636
01640
              template<typename v1, typename v2>
01641
              using lt_t = typename lt_helper<v1, v2>::type;
01642
01646
              template<typename v1, typename v2>
01647
              using gt_t = typename gt_helper<v1, v2>::type;
01648
01652
              template<typename v1, typename v2>
01653
              using div_t = typename div<v1, v2>::q_type;
01654
01658
              template<typename v1, typename v2>
01659
              using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
01660
01664
              template<typename coeff, size_t deg>
01665
              using monomial_t = typename monomial<coeff, deg>::type;
01666
01669
              template<typename v>
01670
              using derive_t = typename derive_helper<v>::type;
01671
01674
              template<typename v>
01675
              using pos_t = typename Ring::template pos_t<typename v::aN>;
01676
01679
              template<tvpename v>
01680
              static constexpr bool pos_v = pos_t<v>::value;
01685
              template<typename v1, typename v2>
01686
              using gcd t = std::conditional t<
01687
                  Ring::is_euclidean_domain,
                  typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
01688
01689
                  void>;
01690
01694
01695
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
01696
01700
              template<tvpename v>
01701
              using inject_ring_t = val<v>;
          };
01703 } // namespace aerobus
01704
01705 // fraction field
01706 namespace aerobus {
01707
         namespace internal {
```

```
template<typename Ring, typename E = void>
01709
              requires IsEuclideanDomain<Ring>
01710
              struct _FractionField {};
01711
01712
              template<typename Ring>
              requires IsEuclideanDomain<Ring>
01713
01714
              struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
01716
                  static constexpr bool is_field = true;
01717
                  static constexpr bool is_euclidean_domain = true;
01718
01719
              private:
01720
                  template<typename val1, typename val2, typename E = void>
01721
                  struct to string helper {};
01722
                  template<typename val1, typename val2>
01723
01724
                  struct to_string_helper <val1, val2,
01725
                      std::enable_if_t<
01726
                      Ring::template eq_t<
                      val2, typename Ring::one
01727
01728
                      >::value
01729
01730
                  > {
01731
                      static std::string func() {
01732
                          return val1::to_string();
01733
01734
                  };
01735
01736
                  template<typename val1, typename val2>
01737
                  struct to_string_helper<val1, val2,
01738
                      std::enable_if_t<
01739
                      !Ring::template eq_t<
01740
                      val2,
01741
                      typename Ring::one
01742
                      >::value
01743
                      >
01744
                  > {
01745
                      static std::string func() {
01746
                          return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
01747
01748
                  };
01749
01750
               public:
01754
                 template<typename val1, typename val2>
01755
                  struct val {
                     using x = val1;
01757
                      using y = val2;
01759
01761
                      using is_zero_t = typename vall::is_zero_t;
01763
                      static constexpr bool is_zero_v = val1::is_zero_t::value;
01764
                      using ring_type = Ring;
01766
01767
                      using enclosing_type = _FractionField<Ring>;
01768
01771
                       static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
01772
01776
                      template<typename valueTvpe>
01777
                      static constexpr valueType get() { return static_cast<valueType>(x::v) /
     static_cast<valueType>(y::v); }
01778
01781
                      static std::string to_string() {
01782
                          return to_string_helper<val1, val2>::func();
01783
01784
01789
                      template<typename valueRing>
01790
                      static constexpr valueRing eval(const valueRing& v) {
01791
                          return x::eval(v) / y::eval(v);
01792
01793
                  };
01794
01796
                  using zero = val<typename Ring::zero, typename Ring::one>;
01798
                  using one = val<typename Ring::one, typename Ring::one>;
01799
01802
                  template<typename v>
01803
                  using inject_t = val<v, typename Ring::one>;
01804
01807
                  template<auto x>
                  using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
     Ring::one>;
01809
01812
                  template<typename v>
                 using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
01813
01814
01816
                 using ring_type = Ring;
01817
               private:
01818
01819
                  template<typename v, typename E = void>
01820
                  struct simplify {};
01821
```

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

```
01908
01909
                                   public:
01910
                                         using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
01911
                                  };
01912
01913
                                  template<tvpename v1, tvpename v2>
                                 struct div<v1, v2, std::enable_if_t<
01914
01915
                                         std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
01916
                                         using type = one;
01917
                                 };
01918
                                 template<typename v1, typename v2>
01919
01920
                                 struct eq {
01921
                                         using type = std::conditional_t<
01922
                                                         std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
01923
                                                         \verb|std::is_same<| typename | simplify_t < v1>::y, | typename | simplify_t < v2>::y>::value, | typename | type
01924
                                                 std::true_type,
01925
                                                std::false_type>;
01926
                                 };
01927
01928
                                  template<typename v1, typename v2, typename E = void>
01929
01930
01931
                                 template<typename v1, typename v2>
struct gt<v1, v2, std::enable_if_t<</pre>
01932
                                       (eq<v1, v2>::type::value)
01933
01934
01935
                                         using type = std::false_type;
01936
                                 };
01937
01938
                                  template<tvpename v1, tvpename v2>
                                  struct gt<v1, v2, std::enable_if_t<
01939
01940
                                        (!eq<v1, v2>::type::value) &&
01941
                                          (!pos<v1>::type::value) && (!pos<v2>::type::value)
01942
01943
                                         using type = typename gt<
01944
                                                typename sub<zero, v1>::type, typename sub<zero, v2>::type
                                         >::type;
01945
01946
                                  };
01947
01948
                                  template<typename v1, typename v2>
                                 struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
01949
01950
01951
                                          (pos<v1>::type::value) && (!pos<v2>::type::value)
01952
01953
                                         using type = std::true_type;
01954
                                  };
01955
01956
                                  template<typename v1, typename v2>
                                 struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
01957
01958
01959
                                          (!pos<v1>::type::value) && (pos<v2>::type::value)
01960
01961
                                         using type = std::false_type;
01962
                                 };
01963
01964
                                  template<typename v1, typename v2>
01965
                                  struct gt<v1, v2, std::enable_if_t<
01966
                                         (!eq<v1, v2>::type::value) &&
01967
                                          (pos<v1>::type::value) && (pos<v2>::type::value)
01968
                                         using type = typename Ring::template gt_t<
01969
01970
                                                typename Ring::template mul_t<v1::x, v2::y>,
01971
                                                 typename Ring::template mul_t<v2::y, v2::x>
01972
01973
                                 } ;
01974
01975
                            public:
01980
                                 template<typename v1, typename v2>
                                 using add_t = typename add<v1, v2>::type;
01982
01987
                                 template<typename v1, typename v2>
01988
                                 using mod_t = zero;
01989
01994
                                 template<typename v1, typename v2>
01995
                                 using gcd_t = v1;
01996
02000
                                 template<typename v1, typename v2>
02001
                                 using sub_t = typename sub<v1, v2>::type;
02002
02006
                                 template<typename v1, typename v2>
02007
                                 using mul_t = typename mul<v1, v2>::type;
02008
02012
                                 template<typename v1, typename v2>
02013
                                 using div_t = typename div<v1, v2>::type;
02014
02018
                                 template<tvpename v1, tvpename v2>
```

```
using eq_t = typename eq<v1, v2>::type;
02020
02024
                  template<typename v1, typename v2>
02025
                  static constexpr bool eq_v = eq<v1, v2>::type::value;
02026
                  template<typename v1, typename v2>
using gt_t = typename gt<v1, v2>::type;
02030
02032
02036
                  template<typename v1, typename v2>
02037
                  static constexpr bool gt_v = gt<v1, v2>::type::value;
02038
02041
                  template<typename v1>
02042
                  using pos t = typename pos<v1>::type;
02043
02046
                  template<typename v>
02047
                  static constexpr bool pos_v = pos_t<v>::value;
02048
              };
02049
02050
              template<typename Ring, typename E = void>
02051
              requires IsEuclideanDomain<Ring>
02052
              struct FractionFieldImpl {};
02053
              // fraction field of a field is the field itself
02054
02055
              template<tvpename Field>
02056
              requires IsEuclideanDomain<Field>
02057
              struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {</pre>
02058
                   using type = Field;
02059
                  template<typename v>
02060
                  using inject_t = v;
02061
              };
02062
02063
              // fraction field of a ring is the actual fraction field
02064
              template<typename Ring>
02065
              requires IsEuclideanDomain<Ring>
02066
              struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {</pre>
02067
                  using type = _FractionField<Ring>;
02068
02069
          } // namespace internal
02070
02074
          template<typename Ring>
02075
          requires IsEuclideanDomain<Ring>
          using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02076
02077 } // namespace aerobus
02078
02079 // short names for common types
02080 namespace aerobus {
02083
          using q32 = FractionField<i32>;
          using fpq32 = FractionField<polynomial<q32>>;
02086
          using q64 = FractionField<i64>;
02089
          using pi64 = polynomial<i64>;
02091
          using pq64 = polynomial < q64>;
02093
02095
          using fpq64 = FractionField<polynomial<q64>>;
02100
          template<typename Ring, typename v1, typename v2>
02101
          using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02102
          template<int64_t p, int64_t q>
02106
          using make_q64_t = typename q64::template simplify_t<
02107
02108
                       typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02109
02113
          template<int32_t p, int32_t q>  
          using make_q32_t = typename q32::template simplify_t<
02114
02115
                       typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>»;
02116
02121
          template<typename Ring, typename v1, typename v2>
02122
          using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02127
          template<typename Ring, typename v1, typename v2>
02128
          \label{eq:constraint} using \ \text{mulfractions\_t} \ = \ \text{typename FractionField} < \text{Ring} > :: \text{template mul\_t} < \text{v1, v2} > ;
        // namespace aerobus
02129 }
02130
02131 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02132 namespace aerobus {
02133
          namespace internal {
02134
              template<typename T, size_t x, typename E = void>
02135
              struct factorial {};
02136
02137
              template<typename T, size_t x>
              struct factorial<T, x, std::enable_if_t<(x > 0)  {
02138
02139
              private:
02140
                  template<typename, size_t, typename>
02141
                  friend struct factorial;
02142
              public:
02143
                  using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
     x - 1>::type>;
02144
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02145
              };
02146
```

```
template<typename T>
              struct factorial<T, 0> {
02148
              public:
02149
02150
                 using type = typename T::one;
                 static constexpr typename T::inner_type value = type::template get<typename</pre>
02151
     T::inner_type>();
02152
             };
02153
          } // namespace internal
02154
02158
          template<typename T, size_t i>
         using factorial_t = typename internal::factorial<T, i>::type;
02159
02160
02164
          template<typename T, size t i>
02165
         inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02166
02167
          namespace internal {
              template<typename T, size_t k, size_t n, typename E = void>
02168
02169
             struct combination_helper {};
02170
02171
              template<typename T, size_t k, size_t n>
02172
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
02173
                 using type = typename FractionField<T>::template mul_t<</pre>
02174
                     typename combination_helper<T, k - 1, n - 1>::type,
02175
                     makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02176
             };
02177
02178
              template<typename T, size_t k, size_t n>
02179
              02180
                 using type = typename combination_helper<T, n - k, n>::type;
02181
02182
02183
              template<typename T, size_t n>
02184
              struct combination_helper<T, 0, n> {
02185
                 using type = typename FractionField<T>::one;
02186
02187
              template<typename T, size_t k, size_t n>
02188
02189
              struct combination {
02190
                 using type = typename internal::combination_helper<T, k, n>::type::x;
02191
                  static constexpr typename T::inner_type value :
02192
                             internal::combination_helper<T, k, n>::type::template get<typename</pre>
     T::inner_type>();
02193
             };
          } // namespace internal
02194
02195
02198
          template<typename T, size_t k, size_t n>
02199
         using combination_t = typename internal::combination<T, k, n>::type;
02200
02205
          template<typename T, size_t k, size_t n>
         inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02206
02207
02208
         namespace internal {
02209
              template<typename T, size_t m>
02210
              struct bernoulli;
02211
02212
              template<typename T, typename accum, size t k, size t m>
02213
              struct bernoulli_helper {
02214
                  using type = typename bernoulli_helper<
02215
02216
                      addfractions_t<T,
02217
                         accum,
                         mulfractions_t<T,</pre>
02218
02219
                             makefraction_t<T,
02220
                                combination_t<T, k, m + 1>,
02221
                                  typename T::one>,
02222
                              typename bernoulli<T, k>::type
02223
02224
                      >,
                      k + 1.
02225
02226
                      m>::type;
02227
02228
02229
              template<typename T, typename accum, size_t m>
              struct bernoulli_helper<T, accum, m, m> {
02230
02231
                 using type = accum;
02232
02233
02234
02235
              template<tvpename T, size_t m>
02236
02237
              struct bernoulli {
02238
                  using type = typename FractionField<T>::template mul_t<</pre>
02239
                     typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02240
                     makefraction_t<T,
02241
                      \label{typename} \mbox{T::template val<static\_cast<typename T::inner\_type>(-1)>,}
02242
                      typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02243
```

```
02244
                  >;
02245
02246
                  template<typename floatType>
                  static constexpr floatType value = type::template get<floatType>();
02247
02248
             };
02249
02250
              template<typename T>
02251
              struct bernoulli<T, 0> {
02252
                 using type = typename FractionField<T>::one;
02253
02254
                  template<typename floatType>
02255
                  static constexpr floatType value = type::template get<floatType>();
02256
              };
02257
         } // namespace internal
02258
02262
          template<typename T, size_t n>
02263
          using bernoulli_t = typename internal::bernoulli<T, n>::type;
02264
02269
          template<typename FloatType, typename T, size_t n >
02270
          inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02271
02272
          // bell numbers
02273
         namespace internal {
              template<typename T, size_t n, typename E = void>
02274
02275
              struct bell_helper;
02276
02277
              template <typename T, size_t n>
02278
              struct bell_helper<T, n, std::enable_if_t<(n > 1)» {
02279
                  template<typename accum, size_t i, size_t stop>
02280
                  struct sum_helper {
02281
                   private:
02282
                      using left = typename T::template mul_t<
02283
                                  combination_t<T, i, n-1>,
02284
                                  typename bell_helper<T, i>::type>;
02285
                      using new_accum = typename T::template add_t<accum, left>;
02286
                   public:
02287
                      using type = typename sum_helper<new_accum, i+1, stop>::type;
02289
02290
                  template<typename accum, size_t stop>
02291
                  struct sum_helper<accum, stop, stop> {
02292
                      using type = accum;
02293
02294
02295
                  using type = typename sum_helper<typename T::zero, 0, n>::type;
02296
             };
02297
02298
              template<typename T>
              struct bell_helper<T, 0> {
02299
02300
                 using type = typename T::one;
02301
              };
02302
02303
              template<typename T>
02304
              struct bell_helper<T, 1> {
02305
                  using type = typename T::one;
02306
02307
         } // namespace internal
02308
02312
          template<typename T, size_t n>
02313
          using bell_t = typename internal::bell_helper<T, n>::type;
02314
         template<typename T, size_t n>
static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
02318
02319
02320
02321
          namespace internal {
02322
              template<typename T, int k, typename E = void>
02323
             struct alternate { };
02324
02325
              template<tvpename T, int k>
              struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
02326
02327
                 using type = typename T::one;
02328
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02329
             };
02330
02331
              template<typename T, int k>
02332
             struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
02333
               using type = typename T::template sub_t<typename T::zero, typename T::one>;
02334
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02335
          } // namespace internal
02336
02337
02340
          template<typename T, int k>
02341
         using alternate_t = typename internal::alternate<T, k>::type;
02342
02343
         namespace internal {
```

```
02344
                template<typename T, int n, int k, typename E = void>
                struct stirling_helper {};
02345
02346
02347
                template<typename T>
                struct stirling_helper<T, 0, 0> {
02348
02349
                    using type = typename T::one;
02350
02351
02352
                template<typename T, int n>
02353
                struct stirling_helper<T, n, 0, std::enable_if_t<(n > 0)  {
02354
                    using type = typename T::zero;
02355
02356
02357
                template<typename T, int n>
02358
                struct stirling_helper<T, 0, n, std::enable_if_t<(n > 0)» {
02359
                    using type = typename T::zero;
02360
               };
02361
02362
                template<typename T, int n, int k>
02363
                struct stirling_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)» {
02364
                    using type = typename T::template sub_t<
02365
                                       typename stirling_helper<T, n-1, k-1>::type,
02366
                                       \label{typename T::template mul_t<} \\
02367
                                            typename T::template inject_constant_t<n-1>,
02368
                                           typename stirling_helper<T, n-1, k>::type
02369
02370
02371
           } // namespace internal
02372
02377
           template<typename T, int n, int k>
02378
           using stirling signed t = typename internal::stirling helper<T, n, k>::type;
02379
02384
           template<typename T, int n, int k>
02385
           using stirling_unsigned_t = abs_t<typename internal::stirling_helper<T, n, k>::type>;
02386
           template<typename T, int n, int k>
static constexpr typename T::inner_type stirling_signed_v = stirling_signed_t<T, n, k>::v;
02391
02392
02393
02394
02399
           template<typename T, int n, int k>
02400
           static constexpr typename T::inner_type stirling_unsigned_v = stirling_unsigned_t<T, n, k>::v;
02401
           template<typename T, size_t k>
inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
02404
02405
02406
02407
           namespace internal {
02408
                template<typename T>
02409
                struct pow_scalar {
02410
                    template<size_t p>
02411
                    static constexpr T func(const T& x) { return p == 0 ? static_cast<T>(1) :
                        p % 2 == 0 ? func < p/2 > (x) * func < p/2 > (x) :
02412
02413
                         x * func<p/2>(x) * func<p/2>(x);
02414
02415
                };
02416
02417
                template<typename T, typename p, size_t n, typename E = void>
                requires IsEuclideanDomain<T>
02418
02419
               struct pow;
02420
02421
                template<typename T, typename p, size_t n>
                struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)» {
    using type = typename T::template mul_t<
02422
02423
02424
                         typename pow<T, p, n/2>::type,
02425
                         typename pow<T, p, n/2>::type
02426
02427
               };
02428
               template<typename T, typename p, size_t n>
struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)» {
    using type = typename T::template mul_t</pre>
02429
02430
02431
02432
02433
                         typename T::template mul_t<
                             typename pow<T, p, n/2>::type, typename pow<T, p, n/2>::type
02434
02435
02436
02437
                    >;
02438
02439
02440
                template<typename T, typename p, size_t n>
                struct pow<T, p, n, std::enable_if_t<n == 0» { using type = typename T::one; };</pre>
02441
           } // namespace internal
02442
02443
           template<typename T, typename p, size_t n>
using pow_t = typename internal::pow<T, p, n>::type;
02448
02449
02450
           template<typename T, typename p, size_t n>
static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
02455
02456
```

```
template<typename T, size_t p>
02458
02459
           static constexpr T pow_scalar(const T& x) { return internal::pow_scalar<T>::template func(x); }
02460
02461
           namespace internal {
               template<typename, template<typename, size_t> typename, class>
02462
02463
               struct make_taylor_impl;
02464
02465
                template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
               struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...» {
    using type = typename polynomial<FractionField<T»::template val<typename coeff_at<T,
02466
02467
      Is>::type...>;
02468
              };
02469
02470
02475
           template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
02476
           using taylor = typename internal::make_taylor_impl<</pre>
02477
02479
               internal::make_index_sequence_reverse<deg + 1>>::type;
02480
02481
           namespace internal {
02482
                template<typename T, size_t i>
02483
                struct exp_coeff {
02484
                    using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02485
02486
02487
                template<typename T, size_t i, typename E = void>
02488
               struct sin_coeff_helper {};
02489
02490
               template<typename T, size_t i>
struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
02491
02492
02493
                };
02494
                template<typename T, size_t i>
02495
               struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
    using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02496
02498
02499
02500
                template<typename T, size_t i>
02501
                struct sin_coeff {
02502
                   using type = typename sin_coeff_helper<T, i>::type;
02503
02504
                template<typename T, size_t i, typename E = void>
02505
02506
                struct sh_coeff_helper {};
02507
02508
                template<tvpename T, size t i>
02509
                struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
                   using type = typename FractionField<T>::zero;
02510
02511
02512
02513
                template<typename T, size_t i>
                struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02514
                   using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02515
02516
02517
02518
                template<typename T, size_t i>
02519
                struct sh_coeff {
                   using type = typename sh_coeff_helper<T, i>::type;
02520
02521
02523
                template<typename T, size_t i, typename E = void>
02524
                struct cos_coeff_helper {};
02525
02526
                template<typename T, size_t i>
                struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
02527
                    using type = typename FractionField<T>::zero;
02528
02530
                template<typename T, size_t i>
02531
                struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02532
02533
02534
02535
02536
                template<typename T, size_t i>
02537
                struct cos_coeff {
02538
                    using type = typename cos_coeff_helper<T, i>::type;
02539
02540
                template<typename T, size_t i, typename E = void>
                struct cosh_coeff_helper {};
02542
02543
02544
                template<typename T, size_t i>
                struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
    using type = typename FractionField<T>::zero;
02545
02546
```

```
02547
              };
02548
02549
              template<typename T, size_t i>
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0\times {
02550
02551
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02552
              };
02553
02554
              template<typename T, size_t i>
02555
              struct cosh_coeff {
02556
                  using type = typename cosh_coeff_helper<T, i>::type;
02557
02558
02559
              template<typename T, size t i>
02560
              struct geom_coeff { using type = typename FractionField<T>::one; };
02561
02562
              template<typename T, size_t i, typename E = void>
02563
02564
              struct atan_coeff_helper;
02565
02566
              template<typename T, size_t i>
02567
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
                  using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;;
02568
02569
              };
02570
02571
              template<typename T, size_t i>
02572
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
                  using type = typename FractionField<T>::zero;
02573
02574
02575
02576
              template<typename T, size_t i>
02577
              struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
02578
02579
              template<typename T, size_t i, typename E = void>
02580
              struct asin_coeff_helper;
02581
              template<typename T, size_t i>
02582
02583
              struct asin coeff helper<T, i, std::enable if t<(i & 1) == 1» {
                  using type = makefraction_t<T,
02584
02585
                       factorial_t<T, i - 1>,
02586
                       typename T::template mul_t<
02587
                           typename T::template val<i>,
02588
                           T::template mul_t<
                               pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
pow<T, factorial_t<T, i / 2>, 2
02589
02590
02591
02592
02593
02594
              };
02595
02596
              template<typename T, size_t i>
02597
              struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02598
                  using type = typename FractionField<T>::zero;
02599
02600
02601
              template<typename T, size_t i>
02602
              struct asin coeff {
02603
                  using type = typename asin_coeff_helper<T, i>::type;
02604
02605
02606
              template<typename T, size_t i>
02607
              struct lnp1 coeff {
                  using type = makefraction_t<T,</pre>
02608
02609
                       alternate_t<T, i + 1>,
                      typename T::template val<i>;;
02610
02611
              };
02612
02613
              template<typename T>
              struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
02614
02615
              template<typename T, size_t i, typename E = void>
02617
              struct asinh_coeff_helper;
02618
02619
              template<typename T, size_t i>
              struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02620
                  using type = makefraction_t<T,
02621
02622
                       typename T::template mul_t<
02623
                           alternate_t<T, i / 2>,
02624
                           factorial_t<T, i - 1>
02625
                       typename T::template mul t<
02626
02627
                           typename T::template mul_t<</pre>
02628
                               typename T::template val<i>,
02629
                               pow_t<T, factorial_t<T, i / 2>, 2>
02630
02631
                           pow_t<T, typename T::template inject_constant_t<4>, i / 2>
02632
02633
                  >;
```

```
02634
               };
02635
02636
               template<typename T, size_t i>
               struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02637
                   using type = typename FractionField<T>::zero;
02638
02639
02640
02641
               template<typename T, size_t i>
02642
               struct asinh_coeff {
02643
                   using type = typename asinh_coeff_helper<T, i>::type;
02644
02645
02646
               template<typename T, size_t i, typename E = void>
02647
               struct atanh_coeff_helper;
02648
               02649
02650
                   // 1/i
02651
02652
                   using type = typename FractionField<T>:: template val<</pre>
02653
                       typename T::one,
02654
                       typename T::template inject_constant_t<i>;;
02655
               };
02656
               template<typename T, size_t i>
struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
02657
02658
                 using type = typename FractionField<T>::zero;
02659
02660
02661
02662
               template<typename T, size_t i>
02663
               struct atanh coeff {
02664
                   using type = typename atanh_coeff_helper<T, i>::type;
02665
02666
02667
               template<typename T, size_t i, typename E = void>
02668
               struct tan_coeff_helper;
02669
02670
               template<typename T, size_t i>
02671
               struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
02672
                  using type = typename FractionField<T>::zero;
02673
02674
02675
               template<typename T, size_t i>
02676
               struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
02677
               private:
02678
                  // 4^((i+1)/2)
                   using _4p = typename FractionField<T>::template inject_t<</pre>
02679
02680
                       pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
                   // 4^((i+1)/2) - 1
02681
                   using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename</pre>
02682
     FractionField<T>::one>;
02683
                      (-1)^{(i-1)/2}
02684
                   using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
02685
                   using dividend = typename FractionField<T>::template mul_t<</pre>
02686
                       altp,
02687
                       FractionField<T>::template mul t<
02688
                       _4p,
FractionField<T>::template mul_t<
02689
02690
                       4pm1.
02691
                       bernoulli_t<T, (i + 1)>
02692
02693
02694
02695
               public:
02696
                  using type = typename FractionField<T>::template div_t<dividend,</pre>
02697
                        typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02698
02699
02700
               template<typename T, size t i>
02701
               struct tan_coeff {
02702
                  using type = typename tan_coeff_helper<T, i>::type;
02703
02704
02705
               template<typename T, size_t i, typename E = void>
02706
               struct tanh_coeff_helper;
02707
02708
               template<typename T, size_t i>
              struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
    using type = typename FractionField<T>::zero;
02709
02710
02711
               };
02712
02713
               template<typename T, size_t i>
               struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
02715
02716
                   using _4p = typename FractionField<T>::template inject_t<</pre>
                   pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename</pre>
02717
02718
      FractionField<T>::one>;
```

```
using dividend =
                       typename FractionField<T>::template mul_t<</pre>
02720
02721
02722
                           typename FractionField<T>::template mul_t<</pre>
02723
                                4pm1,
02724
                               bernoulli_t<T, (i + 1) >>::type;
              public:
02725
02726
                  using type = typename FractionField<T>::template div_t<dividend,</pre>
02727
                      FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02728
              };
02729
02730
              template<typename T, size_t i>
02731
              struct tanh coeff {
02732
                  using type = typename tanh_coeff_helper<T, i>::type;
02733
02734
          } // namespace internal
02735
02739
          template<typename Integers, size_t deg>
using exp = taylor<Integers, internal::exp_coeff, deg>;
02740
02741
02745
          template<typename Integers, size_t deg>
02746
          using expm1 = typename polynomial<FractionField<Integers>>::template sub_t
02747
              exp<Integers, deg>,
02748
              typename polynomial<FractionField<Integers>>::one>;
02749
02753
          template<typename Integers, size_t deg>
02754
          using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
02755
02759
          template<typename Integers, size_t deg>
02760
          using atan = taylor<Integers, internal::atan_coeff, deg>;
02761
02765
          template<typename Integers, size t deg>
02766
          using sin = taylor<Integers, internal::sin_coeff, deg>;
02767
02771
          template<typename Integers, size_t deg>
02772
          using sinh = taylor<Integers, internal::sh_coeff, deg>;
02773
02778
          template<typename Integers, size_t deg>
02779
          using cosh = taylor<Integers, internal::cosh_coeff, deg>;
02780
02785
          template<typename Integers, size_t deg>
02786
          using cos = taylor<Integers, internal::cos_coeff, deg>;
02787
02792
          template<typename Integers, size_t deg>
02793
          using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
02794
02799
          template<typename Integers, size_t deg>
02800
          using asin = taylor<Integers, internal::asin_coeff, deg>;
02801
02806
          template<typename Integers, size t deg>
02807
          using asinh = taylor<Integers, internal::asinh_coeff, deg>;
02808
02813
          template<typename Integers, size_t deg>
02814
          using atanh = taylor<Integers, internal::atanh_coeff, deg>;
02815
02820
          template<typename Integers, size_t deg>
          using tan = taylor<Integers, internal::tan_coeff, deg>;
02821
02822
02827
          template<typename Integers, size_t deg>
02828
          using tanh = taylor<Integers, internal::tanh_coeff, deg>;
02829 }
         // namespace aerobus
02830
02831 // continued fractions
02832 namespace aerobus {
02841
          template<int64_t... values>
02842
          struct ContinuedFraction {};
02843
02846
          template<int64 t a0>
02847
          struct ContinuedFraction<a0> {
              using type = typename q64::template inject_constant_t<a0>;
02851
              static constexpr double val = static_cast<double>(a0);
02852
02853
          template<int64_t a0, int64_t... rest>
02857
          struct ContinuedFraction<a0, rest...> {
    using type = q64::template add_t
02858
02860
02861
                       typename q64::template inject_constant_t<a0>,
02862
                       typename q64::template div_t<
02863
                           typename q64::one,
                           typename ContinuedFraction<rest...>::type
02864
02865
02867
              static constexpr double val = type::template get<double>();
02868
02869
02874
          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>;
02877
          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 =
          using SQRT3_fraction =
02881
          ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 
           // NOLINT
02882 } // namespace aerobus
02883
02884 // known polynomials
02885 namespace aerobus {
                 // CChebyshev
02886
02887
                  namespace internal {
02888
                         template<int kind, size_t deg>
02889
                         struct chebyshev_helper {
02890
                                using type = typename pi64::template sub_t<
                                        typename pi64::template mul_t<
02891
02892
                                                typename pi64::template mul_t<
02893
                                                      pi64::inject_constant_t<2>,
                                                       typename pi64::X>,
02894
02895
                                                typename chebyshev_helper<kind, deg - 1>::type
02896
02897
                                        typename chebyshev_helper<kind, deg - 2>::type
02898
                                >:
02899
                         };
02900
02901
                         template<>
02902
                         struct chebyshev_helper<1, 0> {
                                using type = typename pi64::one;
02903
02904
02905
02906
                         template<>
02907
                         struct chebyshev_helper<1, 1> {
02908
                               using type = typename pi64::X;
02909
                         };
02910
02911
                         template<>
02912
                         struct chebyshev_helper<2, 0> {
02913
                                using type = typename pi64::one;
02914
02915
02916
                         template<>
                         struct chebyshev_helper<2, 1> {
02917
                               using type = typename pi64::template mul_t<
    typename pi64::inject_constant_t<2>,
02918
02919
02920
                                        typename pi64::X>;
02921
02922
                  } // namespace internal
02923
                  // Laguerre
02924
02925
                  namespace internal {
02926
                         template<size_t deg>
02927
                          struct laguerre_helper {
                           private:
02928
                                // Lk = (1 / k) * ((2 * k - 1 - x) * 1km1 - (k - 2) Lkm2) using 1nm2 = typename laguerre_helper<deg - 2>::type;
02929
02930
02931
                                 using lnm1 = typename laguerre_helper<deg - 1>::type;
02932
                                 // -x + 2k-1
02933
                                 using p = typename pq64::template val<
02934
                                      typename q64::template inject_constant_t<-1>,
02935
                                        typename q64::template inject_constant_t<2 \star deg - 1»;
                                 // 1/n
02936
02937
                                 using factor = typename pq64::template inject_ring_t<
02938
                                        q64::val<typename i64::one, typename i64::template inject_constant_t<deg>>;
02939
02940
                           public:
02941
                                 using type = typename pq64::template mul_t <</pre>
02942
                                        factor,
02943
                                        typename pg64::template sub t<
02944
                                                typename pq64::template mul_t<
02945
                                                       p,
02946
02947
                                                typename pq64::template mul_t<
02948
02949
                                                       typename pq64::template inject_constant_t<deg-1>,
02950
                                                       lnm2
02951
02952
02953
                                >;
02954
                         };
02955
02956
                         template<>
                         struct laguerre_helper<0> {
02958
                                using type = typename pq64::one;
02959
02960
02961
                         template<>
02962
                         struct laguerre helper<1> {
```

```
using type = typename pq64::template sub_t<typename pq64::one, typename pq64::X>;
02964
02965
          } // namespace internal
02966
02967
          // Bernstein
02968
          namespace internal {
              template<size_t i, size_t m, typename E = void>
02969
02970
              struct bernstein_helper {};
02971
02972
              template<>
              struct bernstein_helper<0, 0> {
02973
02974
                  using type = typename pi64::one;
02975
              };
02976
02977
               template<size_t i, size_t m>
              02978
02979
                  using type = typename pi64::mul_t<
typename pi64::sub_t<typename pi64::x>,
02980
02981
02982
                           typename bernstein_helper<i, m-1>::type>;
02983
              };
02984
02985
              template<size_t i, size_t m>
              struct bernstein_helper<i, m, std::enable_if_t<  (m > 0) \&\& (i == m) \  \  \{
02986
02987
02988
                   using type = typename pi64::template mul_t<
02989
                           typename pi64::X,
02990
                           typename bernstein_helper<i-1, m-1>::type>;
02991
              };
02992
02993
               template<size t i, size t m>
              struct bernstein_helperi, m, std::enable_if_t<
(m > 0) && (i > 0) && (i < m)» {
02994
02995
02996
                  using type = typename pi64::add_t<
02997
                           typename pi64::mul_t<
02998
                               typename pi64::sub_t<typename pi64::one, typename pi64::X>,
02999
                               typename bernstein_helper<i, m-1>::type>,
03000
                           typename pi64::mul_t<
03001
                               typename pi64::X,
03002
                               typename bernstein_helper<i-1, m-1>::type»;
03003
          } // namespace internal
03004
03005
03006
          namespace known_polynomials {
03008
              enum hermite_kind {
03010
                  probabilist,
03012
                  physicist
03013
              };
03014
          }
03015
03016
          // hermite
03017
          namespace internal {
03018
              template<size_t deg, known_polynomials::hermite_kind kind>
03019
               struct hermite_helper {};
03020
03021
              template<size_t deg>
              struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist> {
03022
               private:
03023
                  using hnm1 = typename hermite_helper<deg - 1,
03024
      known_polynomials::hermite_kind::probabilist>::type;
03025
                  using hnm2 = typename hermite_helper<deg - 2,
      known_polynomials::hermite_kind::probabilist>::type;
03026
03027
03028
                  using type = typename pi64::template sub_t<
03029
                       typename pi64::template mul_t<typename pi64::X, hnm1>,
                       typename pi64::template mul_t<</pre>
03030
03031
                           typename pi64::template inject_constant_t<deg - 1>,
03032
                           hnm2
03033
03034
                  >;
03035
              } ;
03036
03037
              template<size t deg>
03038
              struct hermite helper<deg, known polynomials::hermite kind::physicist> {
03039
03040
                  using hnm1 = typename hermite_helper<deg - 1,
      known_polynomials::hermite_kind::physicist>::type;
03041
                  using hnm2 = typename hermite_helper<deg - 2,
      known_polynomials::hermite_kind::physicist>::type;
03042
               public:
03043
03044
                  using type = typename pi64::template sub_t<
03045
                       // 2X Hn-1
03046
                       typename pi64::template mul_t<
                           typename pi64::val<typename i64::template inject_constant_t<2>,
typename i64::zero>, hnml>,
03047
03048
```

```
03049
03050
                      typename pi64::template mul_t<
03051
                           typename pi64::template inject_constant_t<2*(deg - 1)>,
03052
                          hnm2
03053
03054
                  >;
03055
              };
03056
03057
              template<>
03058
              struct hermite_helper<0, known_polynomials::hermite_kind::probabilist> {
03059
                  using type = typename pi64::one;
03060
03061
03062
              template<>
03063
              struct hermite_helper<1, known_polynomials::hermite_kind::probabilist> {
03064
                 using type = typename pi64::X;
03065
              };
03066
03067
              template<>
03068
              struct hermite_helper<0, known_polynomials::hermite_kind::physicist> {
03069
                  using type = typename pi64::one;
03070
03071
03072
              template<>
03073
              struct hermite_helper<1, known_polynomials::hermite_kind::physicist> {
03074
                  using type = typename pi64::template val<typename i64::template inject_constant_t<2>,
03075
     typename i64::zero>;
03076
              } ;
          } // namespace internal
03077
03078
03079
          // legendre
03080
          namespace internal {
03081
              template<size_t n>
03082
              struct legendre_helper {
03083
               private:
                  // 1/n constant
// (2n-1)/n X
03084
03085
03086
                  using fact_left = typename pq64::monomial_t<make_q64_t<2*n-1, n>, 1>;
03087
03088
                  using fact_right = typename pq64::val<make_q64_t<n-1, n»;
03089
               public:
03090
                  using type = pq64::template sub_t<
03091
                           typename pq64::template mul_t<
                               fact_left,
03092
03093
                               typename legendre_helper<n-1>::type
03094
03095
                           {\tt typename \ pq64::template \ mul\_t<}
03096
                               fact_right,
03097
                               typename legendre helper<n-2>::type
03098
03099
                      >;
03100
              } ;
0.3101
              template<>
03102
03103
              struct legendre helper<0> {
03104
                 using type = typename pq64::one;
03105
03106
03107
              template<>
03108
              struct legendre_helper<1> {
03109
                  using type = typename pq64::X;
03110
03111
          } // namespace internal
03112
03113
          // bernoulli polynomials
03114
          namespace internal {
              template<size t n>
03115
              struct bernoulli_coeff {
03116
                  template<typename T, size_t i>
03117
03118
                  struct inner {
03119
                   private:
03120
                      using F = FractionField<T>;
03121
                   public:
03122
                      using type = typename F::template mul t<
03123
                           typename F::template inject_ring_t<combination_t<T, i, n»,
03124
                           bernoulli_t<T, n-i>
03125
03126
                  };
              };
03127
          } // namespace internal
03128
03129
03131
          namespace known_polynomials {
03138
              template <size_t deg>
03139
              using chebyshev_T = typename internal::chebyshev_helper<1, deg>::type;
0.3140
03147
              template <size t deg>
```

```
using chebyshev_U = typename internal::chebyshev_helper<2, deg>::type;
03149
                                                                     template <size_t deg>
03156
03157
                                                                    using laguerre = typename internal::laguerre_helper<deg>::type;
03158
03165
                                                                     template <size t deg>
03166
                                                                    using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist>::type;
03167
03174
                                                                     template <size_t deg>
03175
                                                                    using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist>::type;
03176
03184
                                                                     template<size t i, size t m>
03185
                                                                    using bernstein = typename internal::bernstein_helper<i, m>::type;
03186
03193
03194
                                                                    using legendre = typename internal::legendre_helper<deg>::type;
03195
03202
                                                                     template<size t deg>
03203
                                                                    using bernoulli = taylor<i64, internal::bernoulli_coeff<deg>::template inner, deg>;
 03204
                                                                // namespace known_polynomials
03205 } // namespace aerobus
03206
03207
03208 #ifdef AEROBUS CONWAY IMPORTS
03209
 03210 // conway polynomials
03211 namespace aerobus {
03215
                                              template<int p, int n>
03216
                                                struct ConwayPolynomial {};
03217
03218 #ifndef DO NOT DOCUMENT
03219
                                                 #define ZPZV ZPZ::template val
                                                  #define POLYV aerobus::polynomial<ZPZ>::template val
03220
                                                 template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
03221
                            POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
03222
                                                 template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
                            POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
03225
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT
03226
                                                 template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
03227
                                                 template<> struct ConwayPolynomial<2, 7> { using ZPZ = aerobus::zpz<2>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
03228
                                               template<> struct ConwayPolynomial<2, 8> { using ZPZ = aerobus::zpz<2>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1»; };
                                               template<> struct ConwayPolynomial<2, 9> { using ZPZ = aerobus::zpz<2>; using type
03229
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03230
                                              template<> struct ConwayPolynomial<2, 10> { using ZPZ = aerobus::zpz<2>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
                            template<> struct ConwayPolynomial<2, 11> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
03231
                             ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<2, 12> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , Z
                                               template<> struct ConwayPolynomial<2, 13> { using ZPZ = aerobus::zpz<2>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
03235
                             ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<2, 16> { using ZPZ = aerobus::zpz<2>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                            ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT
  template<> struct ConwayPolynomial<2, 17> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03237
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<2, 18> { using ZPZ = aerobus::zpz<2>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; }; // NOLINT
                            template<> struct ConwayPolynomial<2, 19> { using ZPZ = aerobus::zpz<<2; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
03239
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
                                                 template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                              // NOLINT
```

```
template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
                              POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
03242
                                                  template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
                              POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2»; }; // NOLINT
03243
                                                    template<> struct ConwayPolynomial<3, 3> { using ZPZ = aerobus::zpz<3>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<1»; };
                                                                                                                                                                                                                                                                            // NOLINT
                                                    template<> struct ConwayPolynomial<3, 4> { using ZPZ = aerobus::zpz<3>; using type =
                              POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; };
                                                                                                                                                                                                                                                                                                                           // NOLINT
03245
                                                  template<> struct ConwayPolynomial<3, 5> { using ZPZ = aerobus::zpz<3>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<3, 6> { using ZPZ = aerobus::zpz<3>; using type =
03246
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2
                                                    template<> struct ConwayPolynomial<3, 7> { using ZPZ = aerobus::zpz<3>; using type
03247
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1»; }; //
03248
                                                 template<> struct ConwayPolynomial<3, 8> { using ZPZ = aerobus::zpz<3>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2
03249
                                                  template<> struct ConwayPolynomial<3, 9> { using ZPZ = aerobus::zpz<3>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; //
                                                    template<> struct ConwayPolynomial<3, 10> { using ZPZ = aerobus::zpz<3>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
                               ZPZV<2»; }; // NOLINT</pre>
03251
                                                  template<> struct ConwayPolynomial<3, 11> { using ZPZ = aerobus::zpz<3>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0); }; // NOLINT template<> struct ConwayPolynomial<3, 12> { using ZPZ = aerobus::zpz<3>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>,
                                ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT</pre>
03253
                                                     template<> struct ConwayPolynomial<3, 13> { using ZPZ = aerobus::zpz<3>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                     template<> struct ConwayPolynomial<3, 14> { using ZPZ = aerobus::zpz<3>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<1 , ZPZV<1
                               \label{eq:condition} \texttt{ZPZV}<\texttt{0>,} \ \texttt{ZPZV}<\texttt{2>,} \ \texttt{ZPZV}<\texttt{1>,} \ \texttt{ZPZV}<\texttt{0>,} \ \texttt{ZPZV}<\texttt{2*;} \ \}; \ \ // \ \texttt{NOLINT}
                              template<> struct ConwayPolynomial<3, 15> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03255
                              ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; // NOLINT
template<> struct ConwayPolynomial<3, 16> { using ZPZ = aerobus::zpz<3>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<3, 17> { using ZPZ = aerobus::zpz<3>; using type =
03257
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               template<> struct ConwayPolynomial<3, 18> { using ZPZ = aerobus::zpz<3>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1
                                ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<0>; };
03259
                                                 template<> struct ConwayPolynomial<3, 19> { using ZPZ = aerobus::zpz<3>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                    template<> struct ConwayPolynomial<3, 20> { using ZPZ = aerobus::zpz<3>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>,
                                ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1
, ZPZV<1
                                 // NOLINT
03261
                                                  template<> struct ConwayPolynomial<5, 1> { using ZPZ = aerobus::zpz<5>; using type =
                              POLYV<ZPZV<1>, ZPZV<3»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<5, 2> { using ZPZ = aerobus::zpz<5>; using type =
                              POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<5, 3> { using ZPZ = aerobus::zpz<5>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
03264
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
03265
                                                    template<> struct ConwayPolynomial<5, 5> { using ZPZ = aerobus::zpz<5>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<3»; }; // NOLINT
03266
                                                 template<> struct ConwayPolynomial<5, 6> { using ZPZ = aerobus::zpz<5>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<5, 7> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>; }; // NOLINT
03267
                                                  template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
03268
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<5, 9> { using ZPZ = aerobus::zpz<5>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<3»; }; //
                               NOLINT
                                                  template<> struct ConwayPolynomial<5, 10> { using ZPZ = aerobus::zpz<5>; using type =
03270
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<1>,
                               ZPZV<2»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<5, 11> { using ZPZ = aerobus::zpz<5>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03272
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4>,
                               ZPZV<3>, ZPZV<2>, ZPZV<2»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<5, 13> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                              ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<3>; }; // NOLINT
   template<> struct ConwayPolynomial<5, 14> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>,
```

```
ZPZV<2>, ZPZV<3>, ZPZV<0>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<5, 15> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>,
                                ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2»; }; // NOLINT
                                                       template<> struct ConwayPolynomial<5, 17> { using ZPZ = aerobus::zpz<5>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                template<> struct ConwayPolynomial<5, 18> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , Z
03278
                                ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<2»; };</pre>
                                                      template<> struct ConwayPolynomial<5, 19> { using ZPZ = aerobus::zpz<5>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<3»; }; //</pre>
                                NOLINT
                               template<> struct ConwayPolynomial<5, 20> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3</pre>
03280
                                 ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<0>, ZPZV<1>, ZPZV<2»; };</pre>
                                // NOLINT
03281
                                                     template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
                               POLYV<ZPZV<1>, ZPZV<4»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
03282
                               POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<7, 3> { using ZPZ = aerobus::zpz<7>; using type =
                              POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type =
03284
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
03285
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT
03286
                                                       template<> struct ConwayPolynomial<7, 6> { using ZPZ = aerobus::zpz<7>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<4>, ZPZV<6>, ZPZV<3»; }; // NOLINT
03287
                                                  template<> struct ConwayPolynomial<7, 7> { using ZPZ = aerobus::zpz<7>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<4»; }; // NOLINT
03288
                                                      template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<3>; }; // NOLINT template<> struct ConwayPolynomial<7, 9> { using ZPZ = aerobus::zpz<7>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 >, ZPZV<6
03290
                                                   template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<3>,
                                ZPZV<3»: 1: // NOLINT
03291
                                                      template<> struct ConwayPolynomial<7, 11> { using ZPZ = aerobus::zpz<7>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                 ZPZV<1>, ZPZV<4»; }; // NOLINT</pre>
03292
                                                   template<> struct ConwayPolynomial<7, 12> { using ZPZ = aerobus::zpz<7>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<5>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<0>, ZPZV<5>, ZPZV<5 , ZPZV<5
                                                      template<> struct ConwayPolynomial<7, 13> { using ZPZ = aerobus::zpz<7>; using type
03293
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<6>, ZPZV<0>, ZPZV<4»; }; // NOLINT</pre>
03294
                                                   template<> struct ConwayPolynomial<7, 14> { using ZPZ = aerobus::zpz<7>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6 , ZPZV<6
                               template<> struct ConwayPolynomial<7, 15> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>,
03295
                                ZPZV<6>, ZPZV<6>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<4»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<7, 16> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
                               ZPZV<3>, ZPZV<4>, ZPZV<1>, ZPZV<6>, ZPZV<2>, ZPZV<4>, ZPZV<3»; }; // NOLINT
  template<> struct ConwayPolynomial<7, 17> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT</pre>
                                                  template<> struct ConwayPolynomial<7, 18> { using ZPZ = aerobus::zpz<7>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<6>, ZPZV<6 , ZPZV<6
                               ZPZV<6>, ZPZV<5>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<2>, ZPZV<2>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<7, 19> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
03299
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<4»; };</pre>
                                                   template<> struct ConwayPolynomial<7, 20> { using ZPZ = aerobus::zpz<7>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<6>, ZPZV<5>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3 , ZPZV<3
                                 // NOLINT
                                                       template<> struct ConwayPolynomial<11, 1> { using ZPZ = aerobus::zpz<11>; using type =
                               POLYV<ZPZV<1>, ZPZV<9»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<11, 2> { using ZPZ = aerobus::zpz<11>; using type =
                              POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
03303
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<9»; }; // NOLINT
template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type =
03304
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<20; }; // NOLINT template<> struct ConwayPolynomial<11, 5> { using ZPZ = aerobus::zpz<11>; using type =
03305
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<0>, ZPZV<9»; }; // NOLINT
                               template<> struct ConwayPolynomial<11, 6> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<6>, ZPZV<7>, ZPZV<2»; }; // NOLINT</pre>
03306
```

```
template<> struct ConwayPolynomial<11, 7> { using ZPZ = aerobus::zpz<11>; using type
                                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<9»; }; // NOLINT
                                                                template<> struct ConwayPolynomial<11, 8> { using ZPZ = aerobus::zpz<11>; using type =
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7
 03309
                                       template<> struct ConwayPolynomial<11, 9> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<8>, ZPZV<8>, ZPZV<8>; }; //
                                                                     template<> struct ConwayPolynomial<11, 10> { using ZPZ = aerobus::zpz<11>; using type
 03310
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<10>, ZPZV<6>, ZPZV<6>,
                                         ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03311
                                                                                                                                                                                                     // NOLINT
                                         ZPZV<10>, ZPZV<9»; };
                                                                   template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<5>, ZPZV<5>,
                                          ZPZV<6>, ZPZV<5>, ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<11, 13> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                                                   template<> struct ConwayPolynomial<11, 14> { using ZPZ = aerobus::zpz<11>; using type
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
                                       ZPZV<4>, ZPZV<6>, ZPZV<6>, ZPZV<10>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<11, 15> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZ
03315
                                       ZPZV<7>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>; ZPZV<6>, ZPZV<6>; ZPZV<6 ; ZPZ
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZ
                                          template<> struct ConwayPolynomial<11, 17> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                                                     template<> struct ConwayPolynomial<11, 18> { using ZPZ = aerobus::zpz<11>; using type
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<8 , ZPZV<8
                                         ZPZV<3>, ZPZV<9>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<9>, ZPZV<8>, ZPZV<2>, ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<11, 19> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
03319
                                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<2>, ZPZV<2>, ZPZV<2</pre>; }; //
                                                                   template<> struct ConwayPolynomial<11, 20> { using ZPZ = aerobus::zpz<11>; using type
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<5>, ZPZV<5 , ZPZV<5
                                         // NOLINT
                                                                  template<> struct ConwayPolynomial<13, 1> { using ZPZ = aerobus::zpz<13>; using type =
03321
                                       POLYV<ZPZV<1>, ZPZV<11»; }; // NOLINT
                                                                     template<> struct ConwayPolynomial<13, 2> { using ZPZ = aerobus::zpz<13>; using type =
                                       POLYV<ZPZV<1>, ZPZV<12>, ZPZV<2»; }; // NOLINT
 03323
                                                              template<> struct ConwayPolynomial<13, 3> { using ZPZ = aerobus::zpz<13>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<13, 4> { using ZPZ = aerobus::zpz<13>; using type =
03324
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2»; }; // NOLINT
                                                                   template<> struct ConwayPolynomial<13, 5> { using ZPZ = aerobus::zpz<13>; using type =
03325
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<11»; }; // NOLINT
 03326
                                                                 template<> struct ConwayPolynomial<13, 6> { using ZPZ = aerobus::zpz<13>; using type =
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<11>, ZPZV<11>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<13, 7> { using ZPZ = aerobus::zpz<13>; using type =
03327
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<11»; }; // NOLINT
                                                                     template<> struct ConwayPolynomial<13, 8> { using ZPZ = aerobus::zpz<13>; using type =
                                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<12>, ZPZV<2>, ZPZV<3>, ZPZV<2»; };
                                                                 template<> struct ConwayPolynomial<13, 9> { using ZPZ = aerobus::zpz<13>, using type
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<8>, ZPZV<12>, ZPZV<12>, ZPZV<11>; };
                                          // NOLINT
                                                                 template<> struct ConwayPolynomial<13, 10> { using ZPZ = aerobus::zpz<13>; using type =
03330
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>,
                                         ZPZV<2»; }; // NOLINT</pre>
                                                              template<> struct ConwayPolynomial<13, 11> { using ZPZ = aerobus::zpz<13>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       ZPZV<3>, ZPZV<11»; }; // NOLINT
    template<> struct ConwayPolynomial<13, 12> { using ZPZ = aerobus::zpz<13>; using type =
03332
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<8>, ZPZV<11>, ZPZV<3>, ZPZV<1>,
                                         ZPZV<1>, ZPZV<4>, ZPZV<2»; }; // NOLINT</pre>
                                                                     template<> struct ConwayPolynomial<13, 13> { using ZPZ = aerobus::zpz<13>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<11»; }; // NOLINT</pre>
                                                                   template<> struct ConwayPolynomial<13, 14> { using ZPZ = aerobus::zpz<13>; using type
03334
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<2»; }; // NOLINT
                                                                template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       ZPZV<2>, ZPZV<11>, ZPZV<10>, ZPZV<11>, ZPZV<8>, ZPZV<11»; }; // NOLINT
    template<> struct ConwayPolynomial<13, 16> { using ZPZ = aerobus::zpz<13>; using type =
POLYV<ZPZV<1>, ZPZV<0>, 
03336
                                       ZPZV<8>, ZPZV<2>, ZPZV<12>, ZPZV<12>, ZPZV<6>, ZPZV<6>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<13, 17> { using ZPZ = aerobus::zpz<13>; using type
                                       POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                                         template<> struct ConwayPolynomial<13, 18> { using ZPZ = aerobus::zpz<13>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<1>, ZPZV<10>, ZPZV<10</pre>
```

```
ZPZV<11>, ZPZV<9>, ZPZV<5>, ZPZV<3>, ZPZV<5>, ZPZV<6>, ZPZV<0>, ZPZV<9>, ZPZV<2»; };</pre>
                             template<> struct ConwayPolynomial<13, 19> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<11»; };</pre>
                              NOLINT
                                                 template<> struct ConwayPolynomial<13, 20> { using ZPZ = aerobus::zpz<13>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<10>, ZPZV<10>
                               ZPZV<9>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<7>, ZPZV<4>, ZPZV<0>, ZPZV<4>, ZPZV<8>, ZPZV<11>, ZPZV<2»; };</pre>
                               // NOLINT
                                                 template<> struct ConwayPolynomial<17, 1> { using ZPZ = aerobus::zpz<17>; using type =
                             POLYV<ZPZV<1>, ZPZV<14»; }; // NOLINT
                                                template<> struct ConwayPolynomial<17, 2> { using ZPZ = aerobus::zpz<17>; using type =
03342
                             POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14»; };
                                                                                                                                                                                                                                                                       // NOLINT
                          template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<10>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<17, 5> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT
03344
03345
                                                 template<> struct ConwayPolynomial<17, 6> { using ZPZ = aerobus::zpz<17>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<10>, ZPZV<3>, ZPZV<3»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<17, 7> { using ZPZ = aerobus::zpz<17>; using type =
03347
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<14»; }; // NOLINT template<> struct ConwayPolynomial<17, 8> { using ZPZ = aerobus::zpz<17>; using type =
03348
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<0>, ZPZV<6>, ZPZV<3»; };
                                                  template<> struct ConwayPolynomial<17, 9> { using ZPZ = aerobus::zpz<17>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<14»; };
                               // NOLINT
                             template<> struct ConwayPolynomial<17, 10> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<12>,
03350
                              ZPZV<3»; }; // NOLINT</pre>
03351
                                                   template<> struct ConwayPolynomial<17, 11> { using ZPZ = aerobus::zpz<17>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<5>, ZPZV<14»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<17, 12> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<16>,
03352
                              ZPZV<14>, ZPZV<9>, ZPZV<3»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<14w; }; // NOLINT
template<> struct ConwayPolynomial<17, 14> { using ZPZ = aerobus::zpz<17>; using type =
03354
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
03355
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<4>, ZPZV<16>, ZPZV<6>, ZPZV<14>, ZPZV<14>, ZPZV<14»; }; // NOLINT</pre>
03356
                                               template<> struct ConwayPolynomial<17, 16> { using ZPZ = aerobus::zpz<17>; using type =
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<13>, ZPZV<1>, ZPZV<1>; // NOLINT template<> struct ConwayPolynomial<17, 17> { using ZPZ = aerobus::zpz<17>; using type =
03357
                              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                               \texttt{ZPZV} < \texttt{0>, ZPZV} < \texttt{14»; }; // \texttt{NOLINT} 
03358
                                              template<> struct ConwayPolynomial<17, 18> { using ZPZ = aerobus::zpz<17>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<16>, ZPZV<7>, ZPZV<1>, ZPZV<3>, ZPZV<3
                             template<> struct ConwayPolynomial<17, 19> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03359
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<14»; }; //</pre>
                             template<> struct ConwayPolynomial<17, 20> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<15>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<15>, ZPZV<15>, ZPZV<15</pre>
03360
                              ZPZV<3»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<19, 1> { using ZPZ = aerobus::zpz<19>; using type =
                             POLYV<ZPZV<1>, ZPZV<17»; }; // NOLINT
03362
                                                template<> struct ConwayPolynomial<19, 2> { using ZPZ = aerobus::zpz<19>; using type =
                             POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2»; }; // NOLINT
                                                template<> struct ConwayPolynomial
19, 3> { using ZPZ = aerobus::zpz<19>; using type =
03363
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type =
03364
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 5> { using ZPZ = aerobus::zpz<19>; using type =
03365
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<19, 6> { using ZPZ = aerobus::zpz<19>; using type =
03366
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<6>, ZPZV<6>, ZPZV<27>, ZPZV<17>, ZPZV<17>, ZPZV<6>, ZPZV<6>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 7> { using ZPZ = aerobus::zpz<19>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17»; };
                                               template<> struct ConwayPolynomial<19, 8> { using ZPZ = aerobus::zpz<19>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<10>, ZPZV<3>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 9> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<14>, ZPZV<16>, ZPZV<16>, ZPZV<17»; };
03369
                              // NOLINT
                           template<> struct ConwayPolynomial<19, 10> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<13>, ZPZV<17>, ZPZV<3>, ZPZV<4>,
                             ZPZV<2»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<19, 11> { using ZPZ = aerobus::zpz<19>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
03371
```

```
ZPZV<8>, ZPZV<17»; };</pre>
                                       template<> struct ConwayPolynomial<19, 12> { using ZPZ = aerobus::zpz<19>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<18>, ZPZV<2>, ZPZV<9>, ZPZV<16>, ZPZV<7>, ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<19, 13> { using ZPZ = aerobus::zpz<19>; using type
03373
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        template<> struct ConwayPolynomial<19, 14> { using ZPZ = aerobus::zpz<19>; using type
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<11>, ZPZV<11>, ZPZV<11>, ZPZV<11>, ZPZV<15>, ZPZV<16>, ZPZV<7>, ZPZV<2»; }; // NOLINT template<>> struct ConwayPolynomial<19, 15> { using ZPZ = aerobus::zpz<19>; using type =
03375
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>,
                       ZPZV<11>, ZPZV<13>, ZPZV<15>, ZPZV<14>, ZPZV<0>, ZPZV<17»; }; // NOLINT
template<> struct ConwayPolynomial<19, 16> { using ZPZ = aerobus::zpz<19>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<13>, ZPZV<0>, ZPZV<15>, ZPZV<9>, ZPZV<6>, ZPZV<14>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<19, 17> { using ZPZ = aerobus::zpz<19>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<17»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<19, 18> { using ZPZ = aerobus::zpz<19>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<1>, ZPZV<7>
                        template<> struct ConwayPolynomial<19, 19> { using ZPZ = aerobus::zpz<19>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1</pre>
03379
                                       template<> struct ConwayPolynomial<19, 20> { using ZPZ = aerobus::zpz<19>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<13>, ZPZV<6>, ZPZV<6 - , ZPZ
                        }; // NOLINT
                                        template<> struct ConwayPolynomial<23, 1> { using ZPZ = aerobus::zpz<23>; using type =
03381
                       POLYV<ZPZV<1>, ZPZV<18»; }; // NOLINT
                                       template<> struct ConwayPolynomial<23, 2> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; }; // NOLINT
03383
                                        template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; }; // NOLINT template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
03384
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT
03385
                                        template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<18»; }; // NOLINT
03386
                                     template<> struct ConwayPolynomial<23, 6> { using ZPZ = aerobus::zpz<23>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<9>, ZPZV<1>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<23, 7> { using ZPZ = aerobus::zpz<23>; using type =
03387
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<18»; }; // NOLINT
                                        template<> struct ConwayPolynomial<23, 8> { using ZPZ = aerobus::zpz<23>; using type :
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<20>, ZPZV<5>, ZPZV<3>, ZPZV<5»; };
03389
                                     template<> struct ConwayPolynomial<23, 9> { using ZPZ = aerobus::zpz<23>; using type :
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<8>, ZPZV<9>, ZPZV<18»; };
                        // NOLINT
                                       template<> struct ConwayPolynomial<23, 10> { using ZPZ = aerobus::zpz<23>; using type
03390
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<1>,
                        ZPZV<5»; }; // NOLINT</pre>
03391
                                     template<> struct ConwayPolynomial<23, 11> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<7>, ZPZV<18»; }; // NOLINT template<> struct ConwayPolynomial<23, 12> { using ZPZ = aerobus::zpz<23>; using type =
03392
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<21>, ZPZV<15>, ZPZV<14>, ZPZV<12>, ZPZV<15, ZPZV<15, ZPZV<16, ZPZV<16, ZPZV<18, Z
                                       template<> struct ConwayPolynomial<23, 13> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       template<> struct ConwayPolynomial<23, 14> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6 , ZPZV<6 ,
                       ZPZV<18>, ZPZV<19>, ZPZV<2>, ZPZV<5>; }; // NOLINT
   template<> struct ConwayPolynomial<23, 15> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<8>, ZPZV<15>, ZPZV<9>, ZPZV<7>, ZPZV<18>, ZPZV<18»; }; // NOLINT template<> struct ConwayPolynomial<23, 16> { using ZPZ = aerobus::zpz<23>; using type =
03396
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>,
                        ZPZV<19>, ZPZV<16>, ZPZV<13>, ZPZV<1>, ZPZV<14>, ZPZV<17>, ZPZV<5»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<23, 17> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                       03398
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<18>, ZPZV<2>, ZPZV<1>,
                        ZPZV<18>, ZPZV<3>, ZPZV<16>, ZPZV<21>, ZPZV<0>, ZPZV<11>, ZPZV<3>, ZPZV<19>, ZPZV<5>; }; // NOLINT
                                     template<> struct ConwayPolynomial<23, 19> { using ZPZ = aerobus::zpz<23>; using type =
03399
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<18»; }; //</pre>
                        NOLINT
                                        template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type =
03400
                      POLYV<ZPZV<1>, ZPZV<27»; }; // NOLINT
                                       template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
                       POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; }; // NOLINT
03402
                                     template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<27»; }; // NOLINT template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
03403
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<2»; };
                                              template<> struct ConwayPolynomial<29, 5> { using ZPZ = aerobus::zpz<29>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<27»; }; // NOLINT
                                           template<> struct ConwayPolynomial<29, 6> { using ZPZ = aerobus::zpz<29>; using type =
03405
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<17>, ZPZV<13>, ZPZV<2»; }; // NOLINT
                                            template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type
03406
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2), ZPZV<2), ZPZV<27»; }; //
03407
                                             template<> struct ConwayPolynomial<29, 8> { using ZPZ = aerobus::zpz<29>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<2»; };
                          template<> struct ConwayPolynomial<29, 9> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<22>, ZPZV<22>, ZPZV<27»; };
03408
                            // NOLINT
                                              template<> struct ConwayPolynomial<29, 10> { using ZPZ = aerobus::zpz<29>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2
                            ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<29, 11> { using ZPZ = aerobus::zpz<29>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<27»; }; // NOLINT
                                              template<> struct ConwayPolynomial<29, 12> { using ZPZ = aerobus::zpz<29>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<39, ZPZV<28>, ZPZV<28>, ZPZV<9>, ZPZV<16>, ZPZV<25>,
                           ZPZV<1>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<29, 13> { using ZPZ = aerobus::zpz<29>; using type =
03412
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              template<> struct ConwayPolynomial<29, 14> { using ZPZ = aerobus::zpz<29>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<3>, ZPZV<14>, ZPZV<10>,
                            ZPZV<21>, ZPZV<18>, ZPZV<27>, ZPZV<5>, ZPZV<2»; }; // NOLINT</pre>
03414
                                             template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::zpz<29>; using type
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           template<> struct ConwayPolynomial<29, 17> { using ZPZ = aerobus::zpz<29>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
03416
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT
template<> struct ConwayPolynomial<29, 18> { using ZPZ = aerobus::zpz<29>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<1>,
                          ZPZV<6>, ZPZV<26>, ZPZV<2>, ZPZV<10>, ZPZV<8>, ZPZV<16>, ZPZV<19>, ZPZV<14>, ZPZV<2*; }; // NOLINT
template<> struct ConwayPolynomial<29, 19> { using ZPZ = aerobus::zpz<29>; using type =
03418
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4</pre>; }; //
                                             template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
                                                                                                                                                          // NOLINT
                          POLYV<ZPZV<1>, ZPZV<28»; };
03420
                                          template<> struct ConwayPolynomial<31, 2> { using ZPZ = aerobus::zpz<31>; using type =
                         POLYV<ZPZV<1>, ZPZV<29>, ZPZV<3»; }; // NOLINT
                                             template<> struct ConwayPolynomial<31, 3> { using ZPZ = aerobus::zpz<31>; using type =
03421
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; // NOLINT template<> struct ConwayPolynomial<31, 4> { using ZPZ = aerobus::zpz<31>; using type =
03422
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3»; }; // NOLINT
03423
                                             template<> struct ConwayPolynomial<31, 5> { using ZPZ = aerobus::zpz<31>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; }; // NOLINT template<> struct ConwayPolynomial<31, 6> { using ZPZ = aerobus::zpz<31>; using type =
03424
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<16>, ZPZV<8>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28»; };
                                          template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<12>, ZPZV<24>, ZPZV<3»; };
                          NOLINT
                                           template<> struct ConwayPolynomial<31, 9> { using ZPZ = aerobus::zpz<31>; using type =
03427
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<20, ZPZV<29>, ZPZV<28»; };
                                          template<> struct ConwayPolynomial<31, 10> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<26>, ZPZV<13>, ZPZV<13>, ZPZV<13>,
                           ZPZV<3»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<31, 11> { using ZPZ = aerobus::zpz<31>; using type =
03429
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<20>, ZPZV<28»; };</pre>
                                                                                                                                      // NOLINT
                                             template<> struct ConwayPolynomial<31, 12> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<28>, ZPZV<2>, ZPZV<9>,
                            ZPZV<25>, ZPZV<12>, ZPZV<3»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<31, 13> { using ZPZ = aerobus::zpz<31>; using type =
03431
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          template<> struct ConwayPolynomial<31, 14> { using ZPZ = aerobus::zpz<31>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<10>, ZPZV<5>, ZPZV<1>,
                           template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type =
03433
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<3
                           ZPZV<29>, ZPZV<12>, ZPZV<13>, ZPZV<23>, ZPZV<25>, ZPZV<28»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<31, 16> { using ZPZ = aerobus::zpz<31>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<24>, ZPZV<26>, ZPZV<28>, ZPZV<11>, ZPZV<19>, ZPZV<27>, ZPZV<23»; }; // NOLINT
    template<> struct ConwayPolynomial<31, 17> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03435
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<28»; };</pre>
                        template<> struct ConwayPolynomial<31, 18> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3 , ZPZV<3 ,
                        ZPZV<2>, ZPZV<7>, ZPZV<12>, ZPZV<11>, ZPZV<25>, ZPZV<25>, ZPZV<10>, ZPZV<6>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<28»; }; //</pre>
                                        template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
03438
                        POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                        template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
03439
                        POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
03440
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT
03441
                                      template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2*; }; // NOLINT
template<> struct ConwayPolynomial<37, 5> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35»; }; // NOLINT
template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
03442
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<4>, ZPZV<30>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<37, 7> { using ZPZ = aerobus::zpz<37>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<7>, ZPZV<35»; };
03445
                                      template<> struct ConwayPolynomial<37, 8> { using ZPZ = aerobus::zpz<37>; using type =
                      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<2P, ZPZV<27>, ZPZV<27>, ZPZV<27>, ZPZV<27>, ZPZV<27>; }; // NOLINT template<> struct ConwayPolynomial<37, 9> { using ZPZ = aerobus::zpz<37>; using type =
03446
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<20>, ZPZV<20>, ZPZV<32>, ZPZV<35»; };
03447
                                       template<> struct ConwayPolynomial<37, 10> { using ZPZ = aerobus::zpz<37>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<11>, ZPZV<4>,
                        ZPZV<2»: }: // NOLINT</pre>
                        template<> struct ConwayPolynomial<37, 11> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                         ZPZV<2>, ZPZV<35»; };</pre>
                                                                                                                    // NOLINT
                                      template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<10>, ZPZV<23>, ZPZV<23>, ZPZV<18>, ZPZV<33>, ZPZV<28+, ZPZV<38+, ZPZV<3
                                        template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type =
03450
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35»; };</pre>
                                                                                                                                                                                               // NOLINT
                                       template<> struct ConwayPolynomial<37, 14> { using ZPZ = aerobus::zpz<37>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35>, ZPZV<35>, ZPZV<1>, ZPZV<32>, ZPZV<16>, ZPZV<1>, ZPZV<9>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<37, 15> { using ZPZ = aerobus::zpz<37>; using type =
03452
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>,
                        ZPZV<28>, ZPZV<27>, ZPZV<13>, ZPZV<34>, ZPZV<35>, ZPZV<35»; ); // NOLINT
template<> struct ConwayPolynomial<37, 17> { using ZPZ = aerobus::zpz<37>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        template<> struct ConwayPolynomial<37, 18> { using ZPZ = aerobus::zpz<37>; using type =
03454
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<20>, ZPZV<2>; }; // NOLINT
                                       template<> struct ConwayPolynomial<37, 19> { using ZPZ = aerobus::zpz<37>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<36>, ZPZV<35»; }; //</pre>
                        NOLINT
                                        template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type =
03456
                        POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                      template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
                        POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6»; }; // NOLINT
03458
                                        template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<35»; }; // NOLINT

template<> struct ConwayPolynomial<41, 4> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<6»; }; // NOLINT
03459
                                         template<> struct ConwayPolynomial<41, 5> { using ZPZ = aerobus::zpz<41>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<14>, ZPZV<35»; }; // NOLINT
03461
                                       template<> struct ConwayPolynomial<41, 6> { using ZPZ = aerobus::zpz<41>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<33>, ZPZV<39>, ZPZV<6>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<41, 7> { using ZPZ = aerobus::zpz<41>; using type =
03462
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT
03463
                                         template<> struct ConwayPolynomial<41, 8> { using ZPZ = aerobus::zpz<41>; using type
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<32>, ZPZV<20>, ZPZV<6>, ZPZV<6»; };
03464
                                      template<> struct ConwayPolynomial<41, 9> { using ZPZ = aerobus::zpz<41>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<5>, ZPZV<35»; };
                        // NOLINT
                                        template<> struct ConwayPolynomial<41, 10> { using ZPZ = aerobus::zpz<41>; using type =
03465
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<8>, ZPZV<80, ZPZV<30>,
                        ZPZV<6»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<41, 11> { using ZPZ = aerobus::zpz<41>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<20>, ZPZV<35»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<41, 12> { using ZPZ = aerobus::zpz<41>; using type
03467
                        POLYYCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<26>, ZPZV<24>, ZPZV<24>, ZPZV<27>, ZPZV<27>, ZPZV<28; }; // NOLINT
                                      template<> struct ConwayPolynomial<41, 13> { using ZPZ = aerobus::zpz<41>; using type =
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
03469
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>,
                             ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>,
                            ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35»; }; // NOLINT
template<> struct ConwayPolynomial<41, 17> { using ZPZ = aerobus::zpz<41>; using type
03471
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35»; };</pre>
                                              template<> struct ConwayPolynomial<41, 18> { using ZPZ = aerobus::zpz<41>; using type =
03472
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<7>, ZPZV<20>,
                            ZPZV<23>, ZPZV<35>, ZPZV<38>, ZPZV<44>, ZPZV<12>, ZPZV<29>, ZPZV<10>, ZPZV<6>, ZPZV<66>; }; // NOLINT template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35»; }; //</pre>
                             NOLINT
03474
                                                 template<> struct ConwayPolynomial<43, 1> { using ZPZ = aerobus::zpz<43>; using type =
                            POLYV<ZPZV<1>, ZPZV<40»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<43, 2> { using ZPZ = aerobus::zpz<43>; using type =
03475
                             POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<43, 3> { using ZPZ = aerobus::zpz<43>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40»; }; // NOLINT template<> struct ConwayPolynomial<43, 4> { using ZPZ = aerobus::zpz<43>; using type =
03477
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<43, 5> { using ZPZ = aerobus::zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<40»; }; // NOLINT
03478
                                                 template<> struct ConwayPolynomial<43, 6> { using ZPZ = aerobus::zpz<43>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<21>, ZPZV<3»; }; // NOLINT
03480
                                               template<> struct ConwayPolynomial<43, 7> { using ZPZ = aerobus::zpz<43>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<7>, ZPZV<7>, ZPZV<40»; }; // NOLINT
03481
                                               template<> struct ConwayPolynomial<43, 8> { using ZPZ = aerobus::zpz<43>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<39>, ZPZV<20>, ZPZV<24>, ZPZV<3*; }; //
                            NOLINT
                                                template<> struct ConwayPolynomial<43, 9> { using ZPZ = aerobus::zpz<43>; using type =
03482
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<39>, ZPZV<1>, ZPZV<40»; };
                              // NOLINT
                                               template<> struct ConwayPolynomial<43, 10> { using ZPZ = aerobus::zpz<43>; using type
03483
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<26>, ZPZV<36>, ZPZV<36>, ZPZV<5>, ZPZV<27>, ZPZV<24>,
                             ZPZV<3»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<7>, ZPZV<40»; }; // NOLINT
   template<> struct ConwayPolynomial<43, 12> { using ZPZ = aerobus::zpz<43>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV
03485
                             ZPZV<23>, ZPZV<38>, ZPZV<3»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<43, 13> { using ZPZ = aerobus::zpz<43>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<40»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<38>, ZPZV<22>, ZPZV<24>,
                            ZPZV<37>, ZPZV<48>, ZPZV<49>, ZPZV<49>, ZPZV<39>, j; // NOLINT template<> struct ConwayPolynomial<43, 15> { using ZPZ = aerobus::zpz<43>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZV<3
                              ZPZV<22>, ZPZV<42>, ZPZV<4>, ZPZV<15>, ZPZV<37>, ZPZV<40»; }; // NOLINT</pre>
                            \label{eq:convayPolynomial} $$ \ 17> \{ using \ ZPZ = aerobus:: zpz<43>; using \ type = POLYV<ZPZV<1>, \ ZPZV<0>, \ ZPZV
03489
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<40»; }; // NOLINT
template<> struct ConwayPolynomial<43, 18> { using ZPZ = aerobus::zpz<43>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5, ZPZV<4>, ZPZV<4 , ZPZV<4 
                            POLIV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZV<3
03491
                             NOLINT
                                                 template<> struct ConwayPolynomial<47, 1> { using ZPZ = aerobus::zpz<47>; using type =
                            POLYV<ZPZV<1>, ZPZV<42»; }; // NOLINT
03493
                                               template<> struct ConwayPolynomial<47, 2> { using ZPZ = aerobus::zpz<47>; using type =
                            POLYV<ZPZV<1>, ZPZV<45>, ZPZV<5»; }; // NOLINT
                                               template<> struct ConwayPolynomial<47, 3> { using ZPZ = aerobus::zpz<47>; using type =
03494
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type =
03495
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<47, 5> { using ZPZ = aerobus::zpz<47>; using type =
03496
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT template<> struct ConwayPolynomial<47, 6> { using ZPZ = aerobus::zpz<47>; using type =
03497
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<35>, ZPZV<9>, ZPZV<41>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<47, 7> { using ZPZ = aerobus::zpz<47>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT
03499
                                               template<> struct ConwayPolynomial<47, 8> { using ZPZ = aerobus::zpz<47>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<19>, ZPZV<3>, ZPZV<5»; };
03500
                            template<> struct ConwayPolynomial<47, 9> { using ZPZ = aerobus::zpz<47>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<42»; };</pre>
                             // NOLINT
                            template<> struct ConwayPolynomial<47, 10> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42>, ZPZV<14>, ZPZV<18>, ZPZV<45>, ZPZV<45>, ZPZV<45>,
                             ZPZV<5»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<47, 11> { using ZPZ = aerobus::zpz<47>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
03502
```

```
ZPZV<6>, ZPZV<42»; };
                          template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<40>, ZPZV<43>, ZPZV<35>, ZPZV<12>, ZPZV<46>,
                           ZPZV<14>, ZPZV<9>, ZPZV<5»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type
03504
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<42»; };</pre>
                                                                                                                                                                                                              // NOLINT
                                             template<> struct ConwayPolynomial<47, 14> { using ZPZ = aerobus::zpz<47>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<30>, ZPZV<30>,
                          ZPZV<17>, ZPZV<24>, ZPZV<9>, ZPZV<32>, ZPZV<5»; }; // NOLINT
  template<> struct ConwayPolynomial<47, 15> { using ZPZ = aerobus::zpz<47>; using type =
03506
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<43>,
                          ZPZV31>, ZPZV414>, ZPZV42>, ZPZV413>, ZPZV417>, ZPZV42»; }; // NOLINT
template<> struct ConwayPolynomial<47, 17> { using ZPZ = aerobus::zpz<47>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 16>, \ \texttt{ZPZV} < 42 \\ \texttt{*}; \ \ // \ \ \texttt{NOLINT} 
                          template<> struct ConwayPolynomial<47, 18> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<41>, ZPZV<41>
                           ZPZV<26>, ZPZV<44>, ZPZV<24>, ZPZV<22>, ZPZV<11>, ZPZV<5>, ZPZV<45>, ZPZV<33>, ZPZV<5»; }; // NOLINT
                                            template<> struct ConwayPolynomial<47, 19> { using ZPZ = aerobus::zpz<47>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<42»; }; //</pre>
                           NOLINT
                                           template<> struct ConwayPolynomial<53, 1> { using ZPZ = aerobus::zpz<53>; using type =
03510
                          POLYV<ZPZV<1>, ZPZV<51»; }; // NOLINT
                                             template<> struct ConwayPolynomial<53, 2> { using ZPZ = aerobus::zpz<53>; using type =
                          POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2»; }; // NOLINT
 03512
                                           template<> struct ConwayPolynomial<53, 3> { using ZPZ = aerobus::zpz<53>; using type =
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
03513
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<38>, ZPZV<38; }; // NOLINT template<> struct ConwayPolynomial<53, 5> { using ZPZ = aerobus::zpz<53>; using type =
 03514
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51»; }; // NOLINT
 03515
                                         template<> struct ConwayPolynomial<53, 6> { using ZPZ = aerobus::zpz<53>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<4>, ZPZV<45>, ZPZV<2»; ); // NOLINT template<> struct ConwayPolynomial<53, 7> { using ZPZ = aerobus::zpz<53>; using type =
 03516
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51»; }; // NOLINT
                                             template<> struct ConwayPolynomial<53, 8> { using ZPZ = aerobus::zpz<53>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<29>, ZPZV<18>, ZPZV<1>, ZPZV<1>, ZPZV<2»; };
 03518
                                          template<> struct ConwayPolynomial<53, 9> { using ZPZ = aerobus::zpz<53>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<5>, ZPZV<51»; };
                           // NOLINT
                          template<> struct ConwayPolynomial<53, 10> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<25, ZPZV<25,
03519
                           ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<53, 11> { using ZPZ = aerobus::zpz<53>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<15>, ZPZV<51»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<53, 12> { using ZPZ = aerobus::zpz<53>; using type =
03521
                          POLYVCZPZVC1>, ZPZVC4), ZPZVC2>, ZPZVC3>, ZPZVC2>, ZPZVC34>, ZPZVC43>, ZPZVC41>, ZPZVC40>, ZPZVC42>, ZPZVC34>, ZPZVC41>, ZPZVC41>, ZPZVC2>; }; // NOLINT
                                           template<> struct ConwayPolynomial<53, 13> { using ZPZ = aerobus::zpz<53>; using type =
                           \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ 
                          ZPZV<0>, ZPZV<52>, ZPZV<28>, ZPZV<51»; }; // NOLINT
template<> struct ConwayPolynomial<53, 14> { using ZPZ = aerobus::zpz<53>; using type =
03523
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<4>, ZPZV<45>, ZPZV<45>, ZPZV<52>,
                            ZPZV<0>, ZPZV<37>, ZPZV<12>, ZPZV<23>, ZPZV<2»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<53, 15> { using ZPZ = aerobus::zpz<53>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51»; }; // NOLINT
    template<> struct ConwayPolynomial<53, 17> { using ZPZ = aerobus::zpz<53>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03525
                            \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 12>, \ \texttt{ZPZV} < 51»; \ // \ \texttt{NOLINT} 
                                             template<> struct ConwayPolynomial<53, 18> { using ZPZ = aerobus::zpz<53>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>,
                           ZPZV<27>, ZPZV<0>, ZPZV<39>, ZPZV<44>, ZPZV<6>, ZPZV<8>, ZPZV<16>, ZPZV<11>, ZPZV<2»; }; // NOLINT</pre>
03527
                          template<> struct ConwayPolynomial<53, 19> { using ZPZ = aerobus::zpz<53>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<51»; }; //</pre>
                           NOLINT
                                             template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
                          POLYV<ZPZV<1>, ZPZV<57»; }; // NOLINT
                                            template<> struct ConwayPolynomial<59, 2> { using ZPZ = aerobus::zpz<59>; using type =
03529
                          POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2»: }; // NOLINT
                                             template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
03530
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; }; // NOLINT
                                          template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
 03531
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<59, 5> { using ZPZ = aerobus::zpz<59>; using type =
03532
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<8>, ZPZV<57»; }; // NOLINT
                                          template<> struct ConwayPolynomial<59, 6> { using ZPZ = aerobus::zpz<59>; using type =
 03533
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18>, ZPZV<38>, ZPZV<0>, ZPZV<2»; };
                         template<> struct ConwayPolynomial<59, 7> { using ZPZ = aerobus::zpz<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>; }; // NOI
                          template<> struct ConwayPolynomial<59, 8> { using ZPZ = aerobus::zpz<59>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<32>, ZPZV<2>, ZPZV<50>, ZPZV<2»; }; //</pre>
03535
```

```
template<> struct ConwayPolynomial<59, 9> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<47>, ZPZV<57»; };
                         // NOLINT
                        template<> struct ConwayPolynomial<59, 10> { using ZPZ = aerobus::zpz<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28>, ZPZV<25>, ZPZV<4>, ZPZV<39>, ZPZV<15>,
03537
                         ZPZV<2»: }: // NOLINT
                                          template<> struct ConwayPolynomial<59, 11> { using ZPZ = aerobus::zpz<59>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<6>, ZPZV<57»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<51>, ZPZV<21>, ZPZV<38>,
ZPZV<8>, ZPZV<1>, ZPZV<2); }; // NOLINT</pre>
03539
                                          template<> struct ConwayPolynomial<59, 13> { using ZPZ = aerobus::zpz<59>; using type
03540
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<59, 14> { using ZPZ = aerobus::zpz<59>; using type =
03541
                        template<> struct ConwayPolynomial<59, 14> { using ZPZ = aerobus::2pZ<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<51>, ZPZV<11>, ZPZV<13>, ZPZV<55>, ZPZV<52>, ZPZV<26>, ZPZV<22>; }; // NOLINT template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>>,
                         ZPZV<24>, ZPZV<23>, ZPZV<13>, ZPZV<39>, ZPZV<58>, ZPZV<57»; }; // NOLINT</pre>
03543
                                          template<> struct ConwayPolynomial<59, 17> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         03544
                                          template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59>; using type
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<38>, ZPZV<27>,
                         ZPZV<11>, ZPZV<14>, ZPZV<7>, ZPZV<44>, ZPZV<16>, ZPZV<47>, ZPZV<34>, ZPZV<32>, ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<59, 19> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<57»; }; //</pre>
                         NOLINT
03546
                                          template<> struct ConwayPolynomial<61, 1> { using ZPZ = aerobus::zpz<61>; using type =
                        POLYV<ZPZV<1>, ZPZV<59»; }; // NOLINT
                                       template<> struct ConwayPolynomial<61, 2> { using ZPZ = aerobus::zpz<61>; using type =
03547
                        POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
03548
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; }; // NOLINT template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<61, 5> { using ZPZ = aerobus::zpz<61>; using type =
03550
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59»; }; // NOLINT
                                          template<> struct ConwayPolynomial<61, 6> { using ZPZ = aerobus::zpz<61>; using type =
03551
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<29>, ZPZV<29>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<61, 7> { using ZPZ = aerobus::zpz<61>; using type
03552
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; }; //
03553
                                          template<> struct ConwayPolynomial<61, 8> { using ZPZ = aerobus::zpz<61>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<1>, ZPZV<56>, ZPZV<2»; };
03554
                                       template<> struct ConwayPolynomial<61, 9> { using ZPZ = aerobus::zpz<61>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<50>, ZPZV<50>, ZPZV<59»; };
                         // NOLINT
03555
                                          template<> struct ConwayPolynomial<61, 10> { using ZPZ = aerobus::zpz<61>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<15>, ZPZV<44>, ZPZV<16>, ZPZV<6>,
                          ZPZV<2»; }; // NOLINT</pre>
03556
                                         template<> struct ConwayPolynomial<61, 11> { using ZPZ = aerobus::zpz<61>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<18>, ZPZV<59»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<61, 12> { using ZPZ = aerobus::zpz<61>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<42>, ZPZV<33>, ZPZV<8>, ZPZV<38>, ZPZV<38>, ZPZV<14>,
                         ZPZV<1>, ZPZV<15>, ZPZV<2»; }; // NOLINT</pre>
03558
                                          template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          template<> struct ConwayPolynomial<61, 14> { using ZPZ = aerobus::zpz<61>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<48>, ZPZV<48>, ZPZV<26>, ZPZV<11>,
                         ZPZV<8>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<2»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<61, 15> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03560
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<59»; }; // NOLINT</pre>
03562
                                       template<> struct ConwayPolynomial<61, 18> { using ZPZ = aerobus::zpz<61>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<36>, ZPZV<36 , ZPZV<37 , ZPZV<37 , ZPZV<37 , ZPZV<38 ,
03563
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; };</pre>
03564
                                          template<> struct ConwayPolynomial<67, 1> { using ZPZ = aerobus::zpz<67>; using type =
                        POLYV<ZPZV<1>, ZPZV<65»; }; // NOLINT
                                          template<> struct ConwayPolynomial<67, 2> { using ZPZ = aerobus::zpz<67>; using type =
03565
                        POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65»; };
                                                                                                                                                                                                                              // NOLINT
03567
                                       template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type =
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<8, ZPZV<54>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<67, 5> { using ZPZ = aerobus::zpz<67>; using type =
03568
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<65»; };
                  template<> struct ConwayPolynomial67, 6> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<63>, ZPZV<49>, ZPZV<55>, ZPZV<2»; }; // NOLINT
                               template<> struct ConwayPolynomial<67, 7> { using ZPZ = aerobus::zpz<67>; using type =
03570
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<67, 8> { using ZPZ = aerobus::zpz<67>; using type =
03571
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<46>, ZPZV<17>, ZPZV<64>, ZPZV<64>, ZPZV<2»; };
                               template<> struct ConwayPolynomial<67, 9> { using ZPZ = aerobus::zpz<67>; using type =
03572
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<49>, ZPZV<55>, ZPZV<65»; };
                     // NOLINT
                               template<> struct ConwayPolynomial<67, 10> { using ZPZ = aerobus::zpz<67>; using type =
03573
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<0>, ZPZV<16>, ZPZV<15, ZPZV<23>,
                    ZPZV<2»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<67, 11> { using ZPZ = aerobus::zpz<67>; using type
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<9>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<67, 12> { using ZPZ = aerobus::zpz<67>; using type =
03575
                   POLYY<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<5>, ZPZV<5>, ZPZV<64>, ZPZV<64>, ZPZV<64>, ZPZV<64>, ZPZV<65>, ZPZV<64>, ZPZV<65>, ZPZV<64>, ZPZV<65>, ZPZV<64>, ZPZV<65>, ZPZV<64>, ZPZV<65>, ZPZV<65>, ZPZV<64>, ZPZV<65>, ZPZV<65>, ZPZV<64>, ZPZV<65>, ZPZV<65>
                   template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                   ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<65»; }; // NOLINT

template<> struct ConwayPolynomial<67, 14> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>,
ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>; }; // NOLINT
03577
                               template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>,
                   ZPZV<52>, ZPZV<41>, ZPZV<20>, ZPZV<21>, ZPZV<46>, ZPZV<65»; }; // NOLINT
    template<> struct ConwayPolynomial<67, 17> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03579
                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<67, 18> { using ZPZ = aerobus::zpz<67>; using type =
03580
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<63>, ZPZV<63>, ZPZV<52>, ZPZV<18>,
                   ZPZV<33>, ZPZV<55>, ZPZV<28>, ZPZV<29>, ZPZV<51>, ZPZV<6>, ZPZV<59>, ZPZV<13>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<67, 19> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
03581
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<65»; }; //</pre>
                                 template<> struct ConwayPolynomial<71, 1> { using ZPZ = aerobus::zpz<71>; using type =
                   POLYV<ZPZV<1>, ZPZV<64»; }; // NOLINT
                                 template<> struct ConwayPolynomial<71, 2> { using ZPZ = aerobus::zpz<71>; using type =
03583
                   POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7»: }: // NOLINT
                                template<> struct ConwayPolynomial71, 3> { using ZPZ = aerobus::zpz71>; using type =
03584
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; }; // NOLINT
03585
                                template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4+, ZPZV<41>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<71, 5> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64»; }; // NOLINT
template<> struct ConwayPolynomial<71, 6> { using ZPZ = aerobus::zpz<71>; using type =
03586
03587
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<13>, ZPZV<29>, ZPZV<7»; }; // NOLINT
                                 template<> struct ConwayPolynomial<71, 7> { using ZPZ = aerobus::zpz<71>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<64»; };
03589
                               template<> struct ConwayPolynomial<71, 8> { using ZPZ = aerobus::zpz<71>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<22>, ZPZV<19>, ZPZV<7»; }; //
                   NOLINT
                                 template<> struct ConwayPolynomial<71, 9> { using ZPZ = aerobus::zpz<71>; using type :
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43>, ZPZV<43>, ZPZV<64»; };
03591
                                 template<> struct ConwayPolynomial<71, 10> { using ZPZ = aerobus::zpz<71>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<17>, ZPZV<26>, ZPZV<1>, ZPZV<40>,
                    ZPZV<7»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<71, 11> { using ZPZ = aerobus::zpz<71>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                    ZPZV<48>, ZPZV<64»; };</pre>
                                                                                                   // NOLINT
                   template<> struct ConwayPolynomial<71, 12> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<28>, ZPZV<29>, ZPZV<55>, ZPZV<21>,
03593
                    ZPZV<58>, ZPZV<23>, ZPZV<7»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<71, 13> { using ZPZ = aerobus::zpz<71>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<27>, ZPZV<64»; }; // NOLINT</pre>
03595
                               template<> struct ConwayPolynomial<71, 15> { using ZPZ = aerobus::zpz<71>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<32>, ZPZV<18>, ZPZV<32>, ZPZV<52>, ZPZV<57>, ZPZV<49>, ZPZV<649; / NOLINT template<> struct ConwayPolynomial<71, 17> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZ
03596
                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<64»; }; // NOLINT</pre>
                   template<> struct ConwayPolynomial<71, 19> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<6+, ZPZV<6+; }; //
                    NOLINT
                                 template<> struct ConwayPolynomial<73, 1> { using ZPZ = aerobus::zpz<73>; using type =
                   POLYV<ZPZV<1>, ZPZV<68»; }; // NOLINT
                               template<> struct ConwayPolynomial<73, 2> { using ZPZ = aerobus::zpz<73>; using type =
                  POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
03600
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<73, 4> { using ZPZ = aerobus::zpz<73>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5», ZPZV<5», ZPZV<5», ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<73, 5> { using ZPZ = aerobus::zpz<73>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<68»; }; // NOLINT
                    template<> struct ConwayPolynomial<73, 6> { using ZPZ = aerobus::zpz<73>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<45>, ZPZV<23>, ZPZV<48>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<73, 7> { using ZPZ = aerobus::zpz<73>; using type =
 03603
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<68»; }; // NOLINT
                                  template<> struct ConwayPolynomial<73, 8> { using ZPZ = aerobus::zpz<73>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<53>, ZPZV<39>, ZPZV<18>, ZPZV<5»; }; //
                     NOLINT
                                  template<> struct ConwayPolynomial<73, 9> { using ZPZ = aerobus::zpz<73>; using type
03606
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<72>, ZPZV<15>, ZPZV<68»; };
 03607
                                  template<> struct ConwayPolynomial<73, 10> { using ZPZ = aerobus::zpz<73>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<23>, ZPZV<33>, ZPZV<32>, ZPZV<69>, ZPZV<5»; }; // NOLINT
                     template<> struct ConwayPolynomial<73, 11> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03608
                      ZPZV<5>, ZPZV<68»; };</pre>
                                                                                                         // NOLINT
                     template<> struct ConwayPolynomial<73, 12> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<52>, ZPZV<26>, ZPZV<20>, ZPZV<46>,
                      ZPZV<29>, ZPZV<25>, ZPZV<5»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<73, 13> { using ZPZ = aerobus::zpz<73>; using type =
03610
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<68»; };</pre>
                                                                                                                                                                        // NOLINT
                                  template<> struct ConwayPolynomial<73, 15> { using ZPZ = aerobus::zpz<73>; using type =
03611
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<10>, ZPZV<33>, ZPZV<57>, ZPZV<57>, ZPZV<62>, ZPZV<68»; }; // NOLINT
    template<> struct ConwayPolynomial<73, 17> { using ZPZ = aerobus::zpz<73>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03612
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<68»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<73, 19> { using ZPZ = aerobus::zpz<73>; using type =
03613
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68»; }; //</pre>
                      NOLINT
03614
                                    template<> struct ConwayPolynomial<79, 1> { using ZPZ = aerobus::zpz<79>; using type =
                     POLYV<ZPZV<1>, ZPZV<76»; }; // NOLINT
                                     template<> struct ConwayPolynomial<79, 2> { using ZPZ = aerobus::zpz<79>; using type =
                      POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3»; }; // NOLINT
03616
                                  template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76»; }; // NOLINT template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
 03617
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3»; };
                                                                                                                                                                                                                             // NOLINT
                                     template<> struct ConwayPolynomial<79, 5> { using ZPZ = aerobus::zpz<79>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<76»; }; // NOLINT
 03619
                                  template<> struct ConwayPolynomial<79, 6> { using ZPZ = aerobus::zpz<79>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<68>, ZPZV<3»; }; // NOLINT
                                    template<> struct ConwayPolynomial<79, 7> { using ZPZ = aerobus::zpz<79>; using type =
03620
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76»; }; // NOLINT
 03621
                                    template<> struct ConwayPolynomial<79, 8> { using ZPZ = aerobus::zpz<79>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<60>, ZPZV<59>, ZPZV<48>, ZPZV<3»; };
                      NOLINT
03622
                                   template<> struct ConwayPolynomial<79, 9> { using ZPZ = aerobus::zpz<79>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<57>, ZPZV<19>, ZPZV<76»; };
                      // NOLINT
                                    template<> struct ConwayPolynomial<79, 10> { using ZPZ = aerobus::zpz<79>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<44>, ZPZV<51>, ZPZV<1>, ZPZV<30>, ZPZV<42>,
                       ZPZV<3»; }; // NOLINT</pre>
03624
                                     template<> struct ConwayPolynomial<79, 11> { using ZPZ = aerobus::zpz<79>; using type
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
03625
                                    template<> struct ConwayPolynomial<79, 12> { using ZPZ = aerobus::zpz<79>; using type
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<45>, ZPZV<52>, ZPZV<7>, ZPZV<40>, ZPZV<40>, ZPZV<59>, ZPZV<59>, ZPZV<50>, ZPZV<59>, ZPZV<50>, ZPZV<3»; }; // NOLINT
03626
                                  template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<78>, ZPZV<4>, ZPZV<76»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type =
03627
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<76»; }; // NOLINT</pre>
03628
                                  template<> struct ConwayPolynomial<79, 19> { using ZPZ = aerobus::zpz<79>; using type
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                      NOLINT
                                    template<> struct ConwayPolynomial<83, 1> { using ZPZ = aerobus::zpz<83>; using type =
                     POLYV<ZPZV<1>, ZPZV<81»; }; // NOLINT
                                  template<> struct ConwayPolynomial<83, 2> { using ZPZ = aerobus::zpz<83>; using type =
                     POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
03631
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT template<> struct ConwayPolynomial<83, 4> { using ZPZ = aerobus::zpz<83>; using type =
 03632
                     POLYVCZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2), ZPZV<2>, ZPZV<3>; // NOLINT template<> struct ConwayPolynomial<83, 5> { using ZPZ = aerobus::zpz<83>; using type =
 03633
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<81»; }; // NOLINT
                     template<> struct ConwayPolynomial<83, 6> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<76>, ZPZV<32>, ZPZV<17>, ZPZV<2»; }; // NOLINT</pre>
 03634
```

```
template<> struct ConwayPolynomial<83, 7> { using ZPZ = aerobus::zpz<83>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT
                                            template<> struct ConwayPolynomial<83, 8> { using ZPZ = aerobus::zpz<83>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<23>, ZPZV<42>, ZPZV<2»; };
                            NOLINT
                                             template<> struct ConwayPolynomial<83, 9> { using ZPZ = aerobus::zpz<83>; using type =
03637
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2+, ZPZV<24, ZPZV<18, ZPZV<81»; };
                                             template<> struct ConwayPolynomial<83, 10> { using ZPZ = aerobus::zpz<83>; using type =
03638
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5 , ZPZV<5
                            ZPZV<2»: }: // NOLINT</pre>
                                            template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type =
03639
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<12>, ZPZV<31>, ZPZV<31>, ZPZV<65>, ZPZV<55>, ZPZV<75>, ZPZV<2»; }; // NOLINT
                                             template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type
03641
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                             template<> struct ConwayPolynomial<83, 17> { using ZPZ = aerobus::zpz<83>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03643
03644
                                              template<> struct ConwayPolynomial<89, 1> { using ZPZ = aerobus::zpz<89>; using type =
                           POLYV<ZPZV<1>, ZPZV<86»; }; // NOLINT
                                             template<> struct ConwayPolynomial<89, 2> { using ZPZ = aerobus::zpz<89>; using type =
03645
                           POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; }; // NOLINT
03646
                                                template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<86>; }; // NOLINT
template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
03647
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<89, 5> { using ZPZ = aerobus::zpz<89>; using type =
03648
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<86»; }; // NOLINT
                                               template<> struct ConwayPolynomial<89, 6> { using ZPZ = aerobus::zpz<89>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<82>, ZPZV<80>, ZPZV<15>, ZPZV<3»; }; // NOLINT
                          template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<86»; }; // NOLINT
                                            template<> struct ConwayPolynomial<89, 8> { using ZPZ = aerobus::zpz<89>; using type =
03651
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<65>, ZPZV<40>, ZPZV<40>, ZPZV<3»; };
                            template<> struct ConwayPolynomial<89, 9> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<86»; };
                             // NOLINT
                            template<> struct ConwayPolynomial<89, 10> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<33>, ZPZV<82>, ZPZV<52>, ZPZV<4>,
03653
                            ZPZV<3»; }; // NOLINT
                                               template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type
03654
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<26>, ZPZV<86»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<89, 12> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<85>, ZPZV<15>, ZPZV<44>, ZPZV<51>, ZPZV<8>,
03655
                            ZPZV<70>, ZPZV<52>, ZPZV<3»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<86»; };</pre>
                                                                                                                                                                                                                             // NOLINT
03657
                                                template<> struct ConwayPolynomial<89, 17> { using ZPZ = aerobus::zpz<89>; using type
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
03658
                                               template<> struct ConwayPolynomial<89, 19> { using ZPZ = aerobus::zpz<89>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0</pre>
                            NOLINT
03659
                                               template<> struct ConwayPolynomial<97, 1> { using ZPZ = aerobus::zpz<97>; using type =
                            POLYV<ZPZV<1>, ZPZV<92»; }; // NOLINT
                                              template<> struct ConwayPolynomial<97, 2> { using ZPZ = aerobus::zpz<97>; using type =
03660
                           POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5»; }; // NOLINT
                                              template<> struct ConwayPolynomial<97, 3> { using ZPZ = aerobus::zpz<97>; using type =
03661
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<92»; }; // NOLINT template<> struct ConwayPolynomial<97, 4> { using ZPZ = aerobus::zpz<97>; using type =
03662
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<97, 5> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<92»; }; // NOLINT
03663
                                             template<> struct ConwayPolynomial<97, 6> { using ZPZ = aerobus::zpz<97>; using type =
03664
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<58>, ZPZV<88>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<97, 7> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>; // NOLINT
03665
                                              template<> struct ConwayPolynomial<97, 8> { using ZPZ = aerobus::zpz<97>; using type =
03666
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<1>, ZPZV<32>, ZPZV<5»; };
                           template<> struct ConwayPolynomial<97, 9> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<5>, ZPZV<5-, 
                            // NOLINT
                            template<> struct ConwayPolynomial<97, 10> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<66>, ZPZV<34>, ZPZV<34>, ZPZV<20>,
03668
```

```
ZPZV<5»; };</pre>
                                                                                        // NOLINT
                                               template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<5>, ZPZV<92»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<97, 12> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<59>, ZPZV<81>, ZPZV<81>, ZPZV<86>,
ZPZV<78>, ZPZV<94>, ZPZV<5»; }; // NOLINT</pre>
03670
                                                template<> struct ConwayPolynomial<97, 13> { using ZPZ = aerobus::zpz<97>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<92»; }; // NOLINT
template<> struct ConwayPolynomial<97, 17> { using ZPZ = aerobus::zpz<97>; using type =
03672
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<52, ZPZV<92w; }; // NOLINT
template<> struct ConwayPolynomial<97, 19> { using ZPZ = aerobus::zpz<97>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<92»; }; //</pre>
                             NOLINT
03674
                                               template<> struct ConwayPolynomial<101, 1> { using ZPZ = aerobus::zpz<101>; using type =
                            POLYV<ZPZV<1>, ZPZV<99»; }; // NOLINT
                                              template<> struct ConwayPolynomial<101, 2> { using ZPZ = aerobus::zpz<101>; using type =
                            POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; }; // NOLINT
03676
                                              template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<99»; }; // NOLINT template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
03677
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<78>, ZPZV<2»; };
                                                                                                                                                                                                                                                                                                         // NOLINT
                                                 template<> struct ConwayPolynomial<101, 5> { using ZPZ = aerobus::zpz<101>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<99»; }; // NOLINT
03679
                                              template<> struct ConwayPolynomial<101, 6> { using ZPZ = aerobus::zpz<101>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<90>, ZPZV<20>, ZPZV<67>, ZPZV<2»; }; // NOLINT
                                              template<> struct ConwayPolynomial<101, 7> { using ZPZ = aerobus::zpz<101>; using type =
03680
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<9>»; }; // NOLINT
03681
                                                template<> struct ConwayPolynomial<101, 8> { using ZPZ = aerobus::zpz<101>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76>, ZPZV<29>, ZPZV<24>, ZPZV<24>, ZPZV<2»; }; //
                           template<> struct ConwayPolynomial<101, 9> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64>, ZPZV<64>, ZPZV<47>, ZPZV<99»; };</pre>
03682
                              // NOLINT
                                                template<> struct ConwayPolynomial<101, 10> { using ZPZ = aerobus::zpz<101>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>>, ZPZV<40>, ZPZV<100>, ZPZV<52>,
                              ZPZV<2»; }; // NOLINT</pre>
03684
                                             template<> struct ConwayPolynomial<101, 11> { using ZPZ = aerobus::zpz<101>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<31>, ZPZV<99»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<101, 12> { using ZPZ = aerobus::zpz<101>; using type =
03685
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<79>, ZPZV<64>, ZPZV<39>, ZPZV<38>, ZPZV<48>, ZPZV<84>, ZPZV<21>, ZPZV<22>; }; // NOLINT
03686
                                             template<> struct ConwayPolynomial<101, 13> { using ZPZ = aerobus::zpz<101>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                               template<> struct ConwayPolynomial<101, 17> { using ZPZ = aerobus::zpz<101>; using type
03687
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<99»; };</pre>
03688
                                              template<> struct ConwayPolynomial<101, 19> { using ZPZ = aerobus::zpz<101>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<99»; }; //</pre>
                             NOLINT
                                                 template<> struct ConwayPolynomial<103, 1> { using ZPZ = aerobus::zpz<103>; using type =
                           POLYV<ZPZV<1>, ZPZV<98»; }; // NOLINT
                                                template<> struct ConwayPolynomial<103, 2> { using ZPZ = aerobus::zpz<103>; using type =
                            POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; }; // NOLINT
                                              template<> struct ConwayPolynomial<103, 3> { using ZPZ = aerobus::zpz<103>; using type =
03691
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<9>»; // NOLINT
template<> struct ConwayPolynomial<103, 4> { using ZPZ = aerobus::zpz<103>; using type =
03692
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<88>, ZPZV<5»; }; // NOLINT
03693
                                             template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<98»; }; // NOLINT
                          template<> struct ConwayPolynomial<103, 6> { using ZPZ = aerobus::zpz<103>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<9>, ZPZV<30>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<103, 7> { using ZPZ = aerobus::zpz<103>; using type =
03694
03695
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98»; };
                                              template<> struct ConwayPolynomial<103, 8> { using ZPZ = aerobus::zpz<103>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<71>, ZPZV<71>, ZPZV<49>, ZPZV<5»; }; //
                            NOLINT
                                              template<> struct ConwayPolynomial<103, 9> { using ZPZ = aerobus::zpz<103>; using type =
03697
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<97>, ZPZV<97>, ZPZV<98»; };
                                              template<> struct ConwayPolynomial<103, 10> { using ZPZ = aerobus::zpz<103>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<101>, ZPZV<86>, ZPZV<101>, ZPZV<10
                             ZPZV<5»; }; // NOLINT</pre>
03699
                                             template<> struct ConwayPolynomial<103, 11> { using ZPZ = aerobus::zpz<103>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              template<> struct ConwayPolynomial<103, 12> { using ZPZ = aerobus::zpz<103>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<74>, ZPZV<23>, ZPZV<94>, ZPZV<94>, ZPZV<81>,
                            ZPZV<29>, ZPZV<88>, ZPZV<5»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<103, 13> { using ZPZ = aerobus::zpz<103>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03701
```

```
ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98»; };</pre>
                                                 template<> struct ConwayPolynomial<103, 17> { using ZPZ = aerobus::zpz<103>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<102>, ZPZV<8>, ZPZV<98»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<103, 19> { using ZPZ = aerobus::zpz<103>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
03703
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2</pre>
                                                 template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
03704
                             POLYV<ZPZV<1>, ZPZV<105»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
03705
                             POLYV<ZPZV<1>, ZPZV<103>, ZPZV<2»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<107, 3> { using ZPZ = aerobus::zpz<107>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<105»; }; // NOLINT
03707
                                              template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<107, 5> { using ZPZ = aerobus::zpz<107>; using type =
03708
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<105»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<107, 6> { using ZPZ = aerobus::zpz<107>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<52>, ZPZV<22>, ZPZV<79>, ZPZV<2»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type =
03710
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<16>, ZPZV<16>, ZPZV<16>, ZPZV<105»; };
                                                                                                                                                                                                                                                                                                                                                                                                                                                                     // NOLINT
                                                template<> struct ConwayPolynomial<107, 8> { using ZPZ = aerobus::zpz<107>; using type =
0.3711
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<24>, ZPZV<95>, ZPZV<95; };
                             NOLINT
                                                 template<> struct ConwayPolynomial<107, 9> { using ZPZ = aerobus::zpz<107>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<36>, ZPZV<105»; };
                              // NOLINT
03713
                                                template<> struct ConwayPolynomial<107, 10> { using ZPZ = aerobus::zpz<107>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<94>, ZPZV<61>, ZPZV<83>, ZPZV<83>, ZPZV<85>,
                              ZPZV<2»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<107, 11> { using ZPZ = aerobus::zpz<107>; using type
                              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<8>, ZPZV<105»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<107, 12> \{ using ZPZ = aerobus::zpz<107>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<37>, ZPZV<48>, ZPZV<6>, ZPZV<6>, ZPZV<61>,
03715
                              ZPZV<42>, ZPZV<57>, ZPZV<2»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105»; };</pre>
                                                                                                                                                                                                                                        // NOLINT
03717
                                               template<> struct ConwayPolynomial<107, 17> { using ZPZ = aerobus::zpz<107>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZ
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<105»; }; //</pre>
                              NOLINT
                                                03719
                             POLYV<ZPZV<1>, ZPZV<103»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<109, 2> { using ZPZ = aerobus::zpz<109>; using type =
03720
                             POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
03722
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<109, 5> { using ZPZ = aerobus::zpz<109>; using type =
03723
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103»; }; // NOLINT
                                               template<> struct ConwayPolynomial<109, 6> { using ZPZ = aerobus::zpz<109>; using type =
03724
                             POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<107>, ZPZY<105, ZPZY<66>, ZPZY<60>, ZPZY<60>; // NOLINT template<> struct ConwayPolynomial<109, 7> { using ZPZ = aerobus::zpz<109>; using type =
03725
                            POLYV-ZPZV-1>, ZPZV-(>, ZPZV-(>, ZPZV-(>), ZPZV-(>, ZPZV-(>), ZPZV-(>), ZPZV-(2), ZPZV-(14), ZPZV-(103); }; // NOLINT template<> struct ConwayPolynomial<109, 8> { using ZPZ = aerobus::zpz<109>; using type =
03726
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<34>, ZPZV<86>, ZPZV<60; };
                                              template<> struct ConwayPolynomial<109, 9> { using ZPZ = aerobus::zpz<109>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<5, ZPZV<9>, ZPZV<9>
                              // NOLINT
                                                 template<> struct ConwayPolynomial<109, 10> { using ZPZ = aerobus::zpz<109>; using type =
03728
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<71>, ZPZV<55>, ZPZV<16>, ZPZV<75>, ZPZV<69>,
                              ZPZV<6»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<109, 11> { using ZPZ = aerobus::zpz<109>; using type :
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<11>, ZPZV<103»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<109, 12> { using ZPZ = aerobus::zpz<109>; using type =
03730
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<50>, ZPZV<53>, ZPZV<37>, ZPZV<65>, ZPZV<65>, ZPZV<103>, ZPZV<28>, ZPZV<69; }; // NOLINT
                                              template<> struct ConwayPolynomial<109, 13> { using ZPZ = aerobus::zpz<109>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              template<> struct ConwayPolynomial<109, 17> { using ZPZ = aerobus::zpz<109>; using type =
03732
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<103»; };</pre>
                                               template<> struct ConwayPolynomial<109, 19> { using ZPZ = aerobus::zpz<109>; using type
                             POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<103»; }; //</pre>
                              NOLINT
03734
                                               template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<110»; };
                                  template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
                   POLYY<ZPZV<1>, ZPZV<101>, ZPZV<3»; ); // NOLINT template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
03736
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<110»; }; // NOLINT template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
03737
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<62>, ZPZV<3»; }; // NOLINT
                                  template<> struct ConwayPolynomial<113, 5> { using ZPZ = aerobus::zpz<113>; using type =
03738
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110»; }; // NOLINT
                   template<> struct ConwayPolynomial<113, 6> { using ZPZ = aerobus::zpz<113>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<50>, ZPZV<30>, ZPZV<71>, ZPZV<3>; }; // NOLINT
template<> struct ConwayPolynomial<113, 7> { using ZPZ = aerobus::zpz<113>; using type =
03739
03740
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<110»; };
                                  template<> struct ConwayPolynomial<113, 8> { using ZPZ = aerobus::zpz<113>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<98>, ZPZV<38>, ZPZV<28>, ZPZV<3»; };
                                 template<> struct ConwayPolynomial<113, 9> { using ZPZ = aerobus::zpz<113>; using type =
03742
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<87>, ZPZV<71>, ZPZV<110»; };
                                  template<> struct ConwayPolynomial<113, 10> { using ZPZ = aerobus::zpz<113>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>,
                     ZPZV<3»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type =
03744
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<3>, ZPZV<110»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<113, 12> { using ZPZ = aerobus::zpz<113>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<23>, ZPZV<62>, ZPZV<4>, ZPZV<4>, ZPZV<56>,
                     ZPZV<10>, ZPZV<27>, ZPZV<3»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<113, 13> { using ZPZ = aerobus::zpz<113>; using type
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   template<> struct ConwayPolynomial<113, 17> { using ZPZ = aerobus::zpz<113>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     template<> struct ConwayPolynomial<113, 19> { using ZPZ = aerobus::zpz<113>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
03748
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<22</pre>
03749
                                  template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
                    POLYV<ZPZV<1>, ZPZV<124»; }; // NOLINT
03750
                                 template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
                    POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3»; }; // NOLINT
                                  template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
03751
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124»; }; // NOLINT
                                  template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<97>, ZPZV<3»; }; // NOLINT
03753
                                template<> struct ConwayPolynomial<127, 5> { using ZPZ = aerobus::zpz<127>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<124»; }; // NOLINT
                                  template<> struct ConwayPolynomial<127, 6> { using ZPZ = aerobus::zpz<127>; using type =
03754
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<84>, ZPZV<115>, ZPZV<82>, ZPZV<3»; }; // NOLINT
                                  template<> struct ConwayPolynomial<127, 7> { using ZPZ = aerobus::zpz<127>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<15>, ZPZV<124»; }; //
03756
                                 template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<104>, ZPZV<55>, ZPZV<8>, ZPZV<3»; }; //
                    NOLINT
03757
                                  template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<119>, ZPZV<126>, ZPZV<124»;
                    }; // NOLINT
                    template<> struct ConwayPolynomial<127, 10> { using ZPZ = aerobus::zpz<127>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<64>, ZPZV<95>, ZPZV<60>, ZPZV<4>,
                     ZPZV<3»: }: // NOLINT
                                 template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<11>, ZPZV<124»; };</pre>
                                                                                                       // NOLINT
                                template<> struct ConwayPolynomial<127, 12> { using ZPZ = aerobus::zpz<127>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<119>, ZPZV<25>, ZPZV<33>, ZPZV<97>, ZPZV<15>,
                     ZPZV<99>, ZPZV<8>, ZPZV<3»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type =
03761
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<124»; };</pre>
                                                                                                                                                                      // NOLINT
                                  template<> struct ConwayPolynomial<127, 17> { using ZPZ = aerobus::zpz<127>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                 ZPZV<0>,
                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<124»; }; // NOLINT
    template<> struct ConwayPolynomial<127, 19> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03763
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30</pre>
                                  template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
                   POLYV<ZPZV<1>, ZPZV<129»; }; // NOLINT
                                  template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
03765
                    POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2»; }; // NOLINT
                                  template<> struct ConwayPolynomial<131, 3> { using ZPZ = aerobus::zpz<131>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<129»; }; // NOLINT template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type =
03767
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<20; }; // NOLINT template<> struct ConwayPolynomial<131, 5> { using ZPZ = aerobus::zpz<131>; using type =
03768
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<129»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<131, 6> { using ZPZ = aerobus::zpz<131>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<44>, ZPZV<22>, ZPZV<22>, ZPZV<22>; }; // NOLINT template<> struct ConwayPolynomial<131, 7> { using ZPZ = aerobus::zpz<131>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<129»; }; // NOLINT
                               template<> struct ConwayPolynomial<131, 8> { using ZPZ = aerobus::zpz<131>; using type =
                   POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<72>, ZPZV<116>, ZPZV<104>, ZPZV<22; }; //
                                template<> struct ConwayPolynomial<131, 9> { using ZPZ = aerobus::zpz<131>; using type =
03772
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<19>, ZPZV<129»; };
                    // NOLINT
03773
                               template<> struct ConwayPolynomial<131, 10> { using ZPZ = aerobus::zpz<131>; using type :
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<97>, ZPZV<97>, ZPZV<97>, ZPZV<44>,
                   ZPZV<2»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<6>, ZPZV<129»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<131, 12> { using ZPZ = aerobus::zpz<131>; using type =
                   POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<50>, ZPZV<122>, ZPZV<40>, ZPZV<83>, ZPZV<125>, ZPZV<28>, ZPZV<28>, ZPZV<29; }; // NOLINT
                               template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<129»; };</pre>
                                                                                                                                                         // NOLINT
                               template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
0.3777
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6</pre>
                                                                                                                                                                                                                                                                             // NOLINT
                                template<> struct ConwayPolynomial<131, 19> { using ZPZ = aerobus::zpz<131>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                         ZPZV<0>,
                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<129»; }; //</pre>
                   NOLINT
03779
                               template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
                  POLYV<ZPZV<1>, ZPZV<134»; }; // NOLINT
03780
                                template<> struct ConwayPolynomial<137, 2> { using ZPZ = aerobus::zpz<137>; using type =
                   POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3»; }; // NOLINT
03781
                              template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
03782
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<3>; }; // NOLINT template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<134»; }; // NOLINT
                 template<> struct ConwayPolynomial<137, 6> { using ZPZ = aerobus::zpz<137>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<116>, ZPZV<102>, ZPZV<3>, ZPZV<3»; }; // NOLINT
03784
                  template<> struct ConwayPolynomial<137, 7> { using ZPZ = aerobus::zpz<137>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13, ZPZV<134»; }; // NOLINT
03785
                               template<> struct ConwayPolynomial<137, 8> { using ZPZ = aerobus::zpz<137>; using type =
03786
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105>, ZPZV<21>, ZPZV<34>, ZPZV<3*);
                   NOLINT
03787
                              template<> struct ConwayPolynomial<137, 9> { using ZPZ = aerobus::zpz<137>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<80>, ZPZV<122>, ZPZV<134»;
                   }; // NOLINT
                                template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type
03788
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<20>, ZPZV<67>, ZPZV<93>, ZPZV<119>,
                   ZPZV<3»; }; // NOLINT</pre>
03789
                              template<> struct ConwayPolynomial<137, 11> { using ZPZ = aerobus::zpz<137>; using type
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<134»; }; // NOLINT
                                template<> struct ConwayPolynomial<137, 12> { using ZPZ = aerobus::zpz<137>; using type
03790
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<61>, ZPZV<40>, ZPZV<40>, ZPZV<12>, ZPZV<36>,
                   ZPZV<135>, ZPZV<61>, ZPZV<3»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<137, 13> { using ZPZ = aerobus::zpz<137>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 17> { using ZPZ = aerobus::zpz<137>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                    \texttt{ZPZV} < \texttt{0>, ZPZV} < \texttt{136>, ZPZV} < \texttt{4>, ZPZV} < \texttt{134*, }; // \texttt{NOLINT} 
                              template<> struct ConwayPolynomial<137, 19> { using ZPZ = aerobus::zpz<137>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<0>, ZPZV<18>, ZPZV<18</pre>
                   NOLINT
                               template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
                  POLYV<ZPZV<1>, ZPZV<137»; }; // NOLINT
                               template<> struct ConwayPolynomial<139, 2> { using ZPZ = aerobus::zpz<139>; using type =
                  POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2»; }; // NOLINT
                                template<> struct ConwayPolynomial<139, 3> { using ZPZ = aerobus::zpz<139>; using type =
03796
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<137»; }; // NOLINT template<> struct ConwayPolynomial<139, 4> { using ZPZ = aerobus::zpz<139>; using type =
03797
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2»; }; // NOLINT
                               template<> struct ConwayPolynomial<139, 5> { using ZPZ = aerobus::zpz<139>; using type =
03798
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<137»; }; // NOLINT
                 template<> struct ConwayPolynomial<139, 6> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<46>, ZPZV<10>, ZPZV<118>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<139, 7> { using ZPZ = aerobus::zpz<139>; using type =
03799
03800
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<137»; };
                               template<> struct ConwayPolynomial<139, 8> { using ZPZ = aerobus::zpz<139>; using type
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>, ZPZV<36>, ZPZV<21>, ZPZV<2»; };
                   template<> struct ConwayPolynomial<139, 9> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<70>, ZPZV<87>, ZPZV<137»; };</pre>
03802
```

```
// NOLINT
03803
                                           template<> struct ConwayPolynomial<139, 10> { using ZPZ = aerobus::zpz<139>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<110>, ZPZV<48>, ZPZV<130>, ZPZV<66>,
                          ZPZV<106>, ZPZV<2»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<139, 11> { using ZPZ = aerobus::zpz<139>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<137»; }; // NOLINT
03804
                                         template<> struct ConwayPolynomial<139, 12> { using ZPZ = aerobus::zpz<139>; using type
03805
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<120>, ZPZV<75>, ZPZV<41>, ZPZV<77>, ZPZV<106>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
                         template<> struct ConwayPolynomia<139, 13> { using ZPZ = aerobus::zpz<139>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 
03806
                          ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<137»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0 , ZPZ
                                           template<> struct ConwayPolynomial<149, 1> { using ZPZ = aerobus::zpz<149>; using type =
                        POLYV<ZPZV<1>, ZPZV<147»; }; // NOLINT
                                         template<> struct ConwayPolynomial<149, 2> { using ZPZ = aerobus::zpz<149>; using type =
03810
                        POLYV<ZPZV<1>, ZPZV<145>, ZPZV<2»; }; // NOLINT
03811
                                           template<> struct ConwayPolynomial<149, 3> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
03812
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<149, 5> { using ZPZ = aerobus::zpz<149>; using type =
03813
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2+, ZPZV<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 6> { using ZPZ = aerobus::zpz<149>; using type =
03814
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<105>, ZPZV<33>, ZPZV<55>, ZPZV<2»; };
                                                                                                                                                                                                                                                                                                                                                                       // NOLINT
                                          template<> struct ConwayPolynomial<149, 7> { using ZPZ = aerobus::zpz<149>; using type =
03815
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<19>, ZPZV<147»; };
03816
                                         template<> struct ConwayPolynomial<149, 8> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<140>, ZPZV<25>, ZPZV<123>, ZPZV<2»; }; //
                         NOLINT
                                         template<> struct ConwayPolynomial<149, 9> { using ZPZ = aerobus::zpz<149>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<146>, ZPZV<20>, ZPZV<147»;
                          }; // NOLINT
03818
                                           template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<74>, ZPZV<42>, ZPZV<148>, ZPZV<143>, ZPZV<51>,
                          ZPZV<2»: : // NOLINT
03819
                                          template<> struct ConwayPolynomial<149, 11> { using ZPZ = aerobus::zpz<149>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<33>, ZPZV<147»; }; // NOLINT</pre>
03820
                                        template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5
                          ZPZV<104>, ZPZV<110>, ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<149, 13> { using ZPZ = aerobus::zpz<149>; using type
03821
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<147»; };</pre>
                                                                                                                                                                                                          // NOLINT
03822
                                        template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type :
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         template<> struct ConwayPolynomial<149, 19> { using ZPZ = aerobus::zpz<149>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03823
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<147»; }; //</pre>
03824
                                          template<> struct ConwayPolynomial<151, 1> { using ZPZ = aerobus::zpz<151>; using type =
                        POLYV<ZPZV<1>, ZPZV<145»; }; // NOLINT
                                         template<> struct ConwayPolynomial<151, 2> { using ZPZ = aerobus::zpz<151>; using type =
03825
                        POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6»; }; // NOLINT
                                           template<> struct ConwayPolynomial<151, 3> { using ZPZ = aerobus::zpz<151>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<145»; }; // NOLINT template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
03827
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<151, 5> { using ZPZ = aerobus::zpz<151>; using type =
03828
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<145»; }; // NOLINT
03829
                                            template<> struct ConwayPolynomial<151, 6> { using ZPZ = aerobus::zpz<151>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<125>, ZPZV<18>, ZPZV<15>, ZPZV<6»; }; // NOLINT
03830
                                        template<> struct ConwayPolynomial<151, 7> { using ZPZ = aerobus::zpz<151>; using type =
                        POLYV<ZPZV<1>, ZPZV<0, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<0>, ZPZV<5, ZPZV<5; 
03831
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<140>, ZPZV<122>, ZPZV<43>, ZPZV<6»; }; //
03832
                                         template<> struct ConwayPolynomial<151, 9> { using ZPZ = aerobus::zpz<151>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<126>, ZPZV<126>, ZPZV<96>, ZPZV<145»;
                          }; // NOLINT
03833
                                         template<> struct ConwayPolynomial<151, 10> { using ZPZ = aerobus::zpz<151>; using type :
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<21>, ZPZV<104>, ZPZV<49, ZPZV<20>, ZPZV<142>,
                         ZPZV<6»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<151, 11> { using ZPZ = aerobus::zpz<151>; using type
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<1>, ZPZV<145»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<151, 12> { using ZPZ = aerobus::zpz<151>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<121>, ZPZV<101>, ZPZV<101>, ZPZV<6>, ZPZV<7>,
03835
```

```
ZPZV<107>, ZPZV<147>, ZPZV<6»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<151, 13> { using ZPZ = aerobus::zpz<151>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        template<> struct ConwayPolynomial<151, 17> { using ZPZ = aerobus::zpz<151>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03837
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<145»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<151, 19> { using ZPZ = aerobus::zpz<151>; using type
03838
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<45»; }; //</pre>
                         NOLINT
                                       template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
03839
                       POLYV<ZPZV<1>, ZPZV<152»; }; // NOLINT
                                         template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
                        POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5»; }; // NOLINT
 03841
                                       template<> struct ConwayPolynomial<157, 3> { using ZPZ = aerobus::zpz<157>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<152»; }; // NOLINT template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
03842
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<136>, ZPZV<5»; }; // NOLINT
                                        template<> struct ConwayPolynomial<157, 5> { using ZPZ = aerobus::zpz<157>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<152»; }; // NOLINT
 03844
                                       \texttt{template<>} \texttt{struct ConwayPolynomial<157, 6> \{ \texttt{using ZPZ = aerobus::zpz<157>; using type = 1.5 } \}
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<130>, ZPZV<43>, ZPZV<144>, ZPZV<55; }; // NOLINT template<> struct ConwayPolynomial<157, 7> { using ZPZ = aerobus::zpz<157>; using type =
03845
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; }; // NOLINT
                                         template<> struct ConwayPolynomial<157, 8> { using ZPZ = aerobus::zpz<157>; using type =
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<9>, ZPZV<40>, ZPZV<40>, ZPZV<153>, ZPZV<5»; };
                        NOLINT
03847
                                        template<> struct ConwayPolynomial<157, 9> { using ZPZ = aerobus::zpz<157>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<114>, ZPZV<52>, ZPZV<152»;
                        }; // NOLINT
03848
                                          template<> struct ConwayPolynomial<157, 10> { using ZPZ = aerobus::zpz<157>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<61>, ZPZV<61>, ZPZV<22>, ZPZV<124>, ZPZV<61>, ZPZV<93>,
                         ZPZV<5»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<157, 11> \{ using ZPZ = aerobus::zpz<157>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV
03849
                         ZPZV<29>, ZPZV<152»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<157, 12> { using ZPZ = aerobus::zpz<157>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110>, ZPZV<72>, ZPZV<137>, ZPZV<43>,
                          ZPZV<152>, ZPZV<57>, ZPZV<5»; }; // NOLINT</pre>
03851
                                       template<> struct ConwayPolynomial<157, 13> { using ZPZ = aerobus::zpz<157>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<156>, ZPZV<5, ZPZV<152»; }; // NOLINT template<> struct ConwayPolynomial<157, 17> { using ZPZ = aerobus::zpz<157>; using type =
03852
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<152»; }; // NOLINT
template<> struct ConwayPolynomial<157, 19> { using ZPZ = aerobus::zpz<157>; using type =
03853
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
                        POLYV<ZPZV<1>, ZPZV<161»; }; // NOLINT
 03855
                                       template<> struct ConwayPolynomial<163, 2> { using ZPZ = aerobus::zpz<163>; using type =
                        POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
03856
                       POLYV<ZPZV<1>, ZPZV<7>, ZPZV<7>, ZPZV<161»; }; // NOLINT template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<163, 5> { using ZPZ = aerobus::zpz<163>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<161»; }; // NOLINT
                       template<> struct ConwayPolynomial<163, 6> { using ZPZ = aerobus::zpz<163>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<25>, ZPZV<156>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<163, 7> { using ZPZ = aerobus::zpz<163>; using type =
03859
 03860
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<161»; }; //
03861
                                       template<> struct ConwayPolynomial<163, 8> { using ZPZ = aerobus::zpz<163>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<132>, ZPZV<83>, ZPZV<6>, ZPZV<2»; };
                        NOLINT
03862
                                       template<> struct ConwayPolynomial<163, 9> { using ZPZ = aerobus::zpz<163>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<16>, ZPZV<162>, ZPZV<121>, ZPZV<161»;
                                       template<> struct ConwayPolynomial<163, 10> { using ZPZ = aerobus::zpz<163>; using type =
03863
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<111>, ZPZV<120>, ZPZV<125>, ZPZV<15>, ZPZV<0>,
                         ZPZV<2»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<163, 11> { using ZPZ = aerobus::zpz<163>; using type =
03864
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<161»; }; // NOLINT
                                      template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<112>, ZPZV<31>, ZPZV<38>, ZPZV<103>, ZPZV<10>, ZPZV<69>, ZPZV<2w; }; // NOLINT
03866
                                       template<> struct ConwayPolynomial<163, 13> { using ZPZ = aerobus::zpz<163>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<161»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<163, 17> { using ZPZ = aerobus::zpz<163>; using type
                        POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<71>, ZPZV<7161»; }; // NOLINT
    template<> struct ConwayPolynomial<163, 19> { using ZPZ = aerobus::zpz<163>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03868
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8161»; }; //</pre>
03869
                                    template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
                       POLYV<ZPZV<1>, ZPZV<162»; }; // NOLINT
 03870
                                      template<> struct ConwayPolynomial<167, 2> { using ZPZ = aerobus::zpz<167>; using type =
                       POLYV<ZPZV<1>, ZPZV<166>, ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<167, 3> { using ZPZ = aerobus::zpz<167>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162»; }; // NOLINT
                                    template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type =
 03872
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<167, 5> { using ZPZ = aerobus::zpz<167>; using type =
03873
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<162»: }; // NOLINT
03874
                                      template<> struct ConwayPolynomial<167, 6> { using ZPZ = aerobus::zpz<167>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<75>, ZPZV<38>, ZPZV<2>, ZPZV<5»; }; // NOLINT
 03875
                                    template<> struct ConwayPolynomial<167, 7> { using ZPZ = aerobus::zpz<167>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162»; }; // NOLINT template<> struct ConwayPolynomial<167, 8> { using ZPZ = aerobus::zpz<167>; using type =
03876
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<149>, ZPZV<56>, ZPZV<113>, ZPZV<55; };
03877
                                      template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<162>, ZPZV<162»;
                       }; // NOLINT
03878
                                      template<> struct ConwayPolynomial<167, 10> { using ZPZ = aerobus::zpz<167>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<68>, ZPZV<109>, ZPZV<143>,
                       ZPZV<148>, ZPZV<5»; };</pre>
                                                                                                                 // NOLINT
                                       template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<24>, ZPZV<162»; }; // NOLINT</pre>
03880
                                       template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<142>, ZPZV<142>, ZPZV<142>, ZPZV<142>, ZPZV<142>, ZPZV<141>, ZPZV<141>, ZPZV<157>, ZPZV<57>, ZPZV<59; }; // NOLINT template<> struct ConwayPolynomial<167, 13> { using ZPZ = aerobus::zpz<167>; using type =
03881
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       template<> struct ConwayPolynomial<167, 17> { using ZPZ = aerobus::zpz<167>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
03882
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<32>, ZPZV<162»; }; // NOLINT
template<> struct ConwayPolynomial<167, 19> { using ZPZ = aerobus::zpz<167>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<14>, ZPZV<162»; }; //</pre>
                       NOLINT
03884
                                      template<> struct ConwayPolynomial<173, 1> { using ZPZ = aerobus::zpz<173>; using type =
                       POLYV<ZPZV<1>, ZPZV<171»; }; // NOLINT
03885
                                      template<> struct ConwayPolynomial<173, 2> { using ZPZ = aerobus::zpz<173>; using type =
                       POLYV<ZPZV<1>, ZPZV<169>, ZPZV<2»; }; // NOLINT
 03886
                                      template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
03887
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<173, 5> { using ZPZ = aerobus::zpz<173>; using type =
 03888
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171»; }; // NOLINT
                                      template<> struct ConwayPolynomial<173, 6> { using ZPZ = aerobus::zpz<173>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<134>, ZPZV<107>, ZPZV<2»; }; // NOLINT
03890
                                     template<> struct ConwayPolynomial<173, 7> { using ZPZ = aerobus::zpz<173>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<171»; }; // NOLINT
                                      template<> struct ConwayPolynomial<173, 8> { using ZPZ = aerobus::zpz<173>; using type =
 03891
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<125>, ZPZV<158>, ZPZV<27>, ZPZV<28; };
                       template<> struct ConwayPolynomial<173, 9> { using ZPZ = aerobus::zpz<173>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<104>, ZPZV<104>, ZPZV<104>, ZPZV<171»;
                       }; // NOLINT
                                      template<> struct ConwayPolynomial<173, 10> { using ZPZ = aerobus::zpz<173>; using type =
03893
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<156>, ZPZV<164>, ZPZV<48>, ZPZV<106>,
                                                                                                             // NOLINT
                       ZPZV<58>, ZPZV<2»; };</pre>
                                    template<> struct ConwayPolynomial<173, 11> { using ZPZ = aerobus::zpz<173>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<12>, ZPZV<171»; }; // NOLINT
                                     template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type
03895
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<64>, ZPZV<46>, ZPZV<166>, ZPZV<10>,
                       ZPZV<159>, ZPZV<22>, ZPZV<2»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<173, 13> { using ZPZ = aerobus::zpz<173>; using type :
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<171»; }; // NOLINT
template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type
03897
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<171»; };</pre>
                                    template<> struct ConwayPolynomial<173, 19> { using ZPZ = aerobus::zpz<173>; using type
 03898
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6</pre>
                       NOLINT
03899
                                      template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
                       POLYV<ZPZV<1>, ZPZV<177»; };
                                                                                                                                       // NOLINT
                                      template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
                       POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2»; }; // NOLINT
 03901
                                    template<> struct ConwayPolynomial<179, 3> { using ZPZ = aerobus::zpz<179>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<177»; }; // NOLINT template<> struct ConwayPolynomial<179, 4> { using ZPZ = aerobus::zpz<179>; using type =
 03902
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<2»; };
                                          template<> struct ConwayPolynomial<179, 5> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<177»; }; // NOLINT
                                         template<> struct ConwayPolynomial<179, 6> { using ZPZ = aerobus::zpz<179>; using type =
 03904
                          \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 7>, \ \texttt{ZPZV} < 91>, \ \texttt{ZPZV} < 55>, \ \texttt{ZPZV} < 109>, \ \texttt{ZPZV} < 2»; \ \}; \ \ // \ \texttt{NOLINT} 
03905
                                         template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 >, ZP
                                          template<> struct ConwayPolynomial<179, 8> { using ZPZ = aerobus::zpz<179>; using type
 03906
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<144>, ZPZV<73>, ZPZV<2»; };
03907
                                        template<> struct ConwayPolynomial<179, 9> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<40>, ZPZV<64>, ZPZV<64>, ZPZV<177»; };
                         // NOLINT
                                           template<> struct ConwayPolynomial<179, 10> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<115>, ZPZV<71>, ZPZV<150>, ZPZV<49>, ZPZV<87>,
                          ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<179, 11> { using ZPZ = aerobus::zpz<179>; using type =
03909
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           template<> struct ConwayPolynomial<179, 12> { using ZPZ = aerobus::zpz<179>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<103>, ZPZV<83>, ZPZV<43>, ZPZV<76>, ZPZV<8>, ZPZV<177>, ZPZV<1>, ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<179, 13> { using ZPZ = aerobus::zpz<179>; using type =
03911
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<177»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<179,
                                                                                                                                                                                                              17> { using ZPZ = aerobus::zpz<179>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                       // NOLINT
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177»; };</pre>
                                          template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type =
03913
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177»; }; //</pre>
                         NOLINT
                                          template<> struct ConwayPolynomial<181, 1> { using ZPZ = aerobus::zpz<181>; using type =
 03914
                         POLYV<ZPZV<1>, ZPZV<179»; }; // NOLINT
 03915
                                          template<> struct ConwayPolynomial<181, 2> { using ZPZ = aerobus::zpz<181>; using type =
                         POLYV<ZPZV<1>, ZPZV<177>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
03916
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179»; }; // NOLINT
 03917
                                           template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<105>, ZPZV<2w; }; // NOLINT template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
 03918
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2), ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>; ; // NOLINT template<> struct ConwayPolynomial<181, 6> { using ZPZ = aerobus::zpz<181>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<163>, ZPZV<169>, ZPZV<2»; }; // NOLINT
 03919
                                           template<> struct ConwayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<179»; };
 03921
                                       template<> struct ConwayPolynomial<181, 8> { using ZPZ = aerobus::zpz<181>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<108>, ZPZV<22>, ZPZV<149>, ZPZV<2»; }; //
                         NOLINT
                                         template<> struct ConwayPolynomial<181, 9> { using ZPZ = aerobus::zpz<181>; using type =
03922
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<168>, ZPZV<179»;
                         }; // NOLINT
03923
                                        template<> struct ConwayPolynomial<181, 10> { using ZPZ = aerobus::zpz<181>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<154>, ZPZV<104>, ZPZV<94>, ZPZV<57>, ZPZV<88>,
                         ZPZV<2»: }: // NOLINT</pre>
03924
                                          template<> struct ConwayPolynomial<181, 11> { using ZPZ = aerobus::zpz<181>; using type :
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                           // NOLINT
                         ZPZV<24>, ZPZV<179»; };</pre>
                         template<> struct ConwayPolynomial<181, 12> { using ZPZ = aerobus::zpz<181>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<141>, ZPZV<45>, ZPZV<122>,
                         ZPZV<175>, ZPZV<12>, ZPZV<10>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<181, 13> { using ZPZ = aerobus::zpz<181>; using type =
03926
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        template<> struct ConwayPolynomial<181, 17> { using ZPZ = aerobus::zpz<181>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<179»; }; // NOLINT
template<> struct ConwayPolynomial<181, 19> { using ZPZ = aerobus::zpz<181>; using type =
03928
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<179»; }; //</pre>
                                        template<> struct ConwayPolynomial<191, 1> { using ZPZ = aerobus::zpz<191>; using type =
                         POLYV<ZPZV<1>, ZPZV<172»; }; // NOLINT
                                          template<> struct ConwayPolynomial<191, 2> { using ZPZ = aerobus::zpz<191>; using type =
03930
                         POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19»; }; // NOLINT
                                           template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172»; }; // NOLINT
 03932
                                        template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; }; // NOLINT
                                         template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
 03933
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<172»; }; // NOLINT
 03934
                                          template<> struct ConwayPolynomial<191, 6> { using ZPZ = aerobus::zpz<191>; using type =
                        POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<110>, ZPZV<10>, ZPZV<19»; }; // NOLINT template<> struct ConwayPolynomial<191, 7> { using ZPZ = aerobus::zpz<191>; using type =
 03935
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<14>, ZPZV<172»; }; // NOLINT template<> struct ConwayPolynomial<191, 8> { using ZPZ = aerobus::zpz<191>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<164>, ZPZV<139>, ZPZV<171>, ZPZV<19»; }; //
 03936
```

```
NOLINT
03937
                                       template<> struct ConwayPolynomial<191, 9> { using ZPZ = aerobus::zpz<191>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<62>, ZPZV<124>, ZPZV<172»;
                        }; // NOLINT
03938
                                          template<> struct ConwayPolynomial<191, 10> { using ZPZ = aerobus::zpz<191>; using type =
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<173>, ZPZV<14>, ZPZV<156>, ZPZV<19»; }; // NOLINT
                                         template<> struct ConwayPolynomial<191, 11> { using ZPZ = aerobus::zpz<191>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<6>, ZPZV<172»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<191, 12> { using ZPZ = aerobus::zpz<191>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<168>, ZPZV<25>, ZPZV<49>, ZPZV<90>,
03940
                         ZPZV<7>, ZPZV<151>, ZPZV<19»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<191, 13> { using ZPZ = aerobus::zpz<191>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         template<> struct ConwayPolynomial<191, 17> { using ZPZ = aerobus::zpz<191>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<172»; };</pre>
                                                                                                                                                                                                                                                                                                                                                     // NOLINT
                                       template<> struct ConwayPolynomial<191, 19> { using ZPZ = aerobus::zpz<191>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<190>, ZPZV<2>, ZPZV<172»; }; //</pre>
                         NOLINT
                                       template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
03944
                        POLYV<ZPZV<1>, ZPZV<188»; }; // NOLINT
                                         template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
                        POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5»; }; // NOLINT
03946
                                       template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
03947
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<66, ZPZV<148>, ZPZV<58; }; // NOLINT template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
03948
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<188»; }; // NOLINT
03949
                                      template<> struct ConwayPolynomial<193, 6> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<8>, ZPZV<172>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<193, 7> { using ZPZ = aerobus::zpz<193>; using type =
03950
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<88, ZPZV<188»; }; // NOLINT
                                       template<> struct ConwayPolynomial<193, 8> { using ZPZ = aerobus::zpz<193>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<145>, ZPZV<34>, ZPZV<154>, ZPZV<15*; //
03952
                                      template<> struct ConwayPolynomial<193, 9> { using ZPZ = aerobus::zpz<193>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<168>, ZPZV<17>, ZPZV<188»;
                         }; // NOLINT
03953
                                          template<> struct ConwayPolynomial<193, 10> { using ZPZ = aerobus::zpz<193>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<51>, ZPZV<77>, ZPZV<0>, ZPZV<89>,
                         ZPZV<5»; }; // NOLINT</pre>
03954
                                       template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<1>, ZPZV<188»: }: // NOLINT</pre>
                                         template<> struct ConwayPolynomial<193, 12> { using ZPZ = aerobus::zpz<193>; using type =
03955
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<52>, ZPZV<135>, ZPZV<155>,
                         ZPZV<90>, ZPZV<46>, ZPZV<28>, ZPZV<5»; }; // NOLINT</pre>
03956
                                      template<> struct ConwayPolynomial<193,
                                                                                                                                                                                                            13> { using ZPZ = aerobus::zpz<193>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        template<> struct ConwayPolynomial<193,
                                                                                                                                                                                                            17> { using ZPZ = aerobus::zpz<193>; using type =
03957
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<188»; };</pre>
                                       template<> struct ConwayPolynomial<193, 19> { using ZPZ = aerobus::zpz<193>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                          \texttt{ZPZV} < \texttt{0>, ZPZV} < \texttt{0
                         NOLINT
03959
                                         template<> struct ConwayPolynomial<197, 1> { using ZPZ = aerobus::zpz<197>; using type =
                        POLYV<ZPZV<1>, ZPZV<195»; }; // NOLINT
                                      template<> struct ConwayPolynomial<197, 2> { using ZPZ = aerobus::zpz<197>; using type =
                       POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2»; }; // NOLINT
03961
                                        template<> struct ConwayPolynomial<197, 3> { using ZPZ = aerobus::zpz<197>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<195»; ); // NOLINT template<> struct ConwayPolynomial<197, 4> { using ZPZ = aerobus::zpz<197>; using type =
03962
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<124>, ZPZV<2»; };
                                                                                                                                                                                                                                                                     // NOLINT
                                       template<> struct ConwayPolynomial<197, 5> { using ZPZ = aerobus::zpz<197>; using type =
03963
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195»; }; // NOLINT
03964
                                        template<> struct ConwayPolynomial<197, 6> { using ZPZ = aerobus::zpz<197>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<124>, ZPZV<79>, ZPZV<173>, ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<197,
                                                                                                                                                                                                            7> { using ZPZ = aerobus::zpz<197>; using type
03965
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<195»; }; //
                                      template<> struct ConwayPolynomial<197, 8> { using ZPZ = aerobus::zpz<197>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<176>, ZPZV<96>, ZPZV<29>, ZPZV<2»; };
                        NOLINT
03967
                                       template<> struct ConwayPolynomial<197, 9> { using ZPZ = aerobus::zpz<197>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<127>, ZPZV<8>, ZPZV<195»;
                      }; // NOLINT
template<> struct ConwayPolynomial<197, 10> { using ZPZ = aerobus::zpz<197>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<137>, ZPZV<8>, ZPZV<42>,
                                       template<> struct ConwayPolynomial<197, 11> { using ZPZ = aerobus::zpz<197>; using type =
03969
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
```

```
ZPZV<14>, ZPZV<195»; };
                                              template<> struct ConwayPolynomial<197, 12> { using ZPZ = aerobus::zpz<197>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>,
                            ZPZV<90>, ZPZV<163>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<197, 13> { using ZPZ = aerobus::zpz<197>; using type :
03971
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                               template<> struct ConwayPolynomial<197, 17> { using ZPZ = aerobus::zpz<197>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<195»; }; // NOLINT
    template<> struct ConwayPolynomial<197, 19> { using ZPZ = aerobus::zpz<197>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03973
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6</pre>, ZPZV<195»; }; //</pre>
03974
                                            template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
                          POLYV<ZPZV<1>, ZPZV<196»; }; // NOLINT template<> struct ConwayPolynomial<199, 2> { using ZPZ = aerobus::zpz<199>; using type =
03975
                          POLYV<ZPZV<1>, ZPZV<193>, ZPZV<3>; ; // NOLINT template<> struct ConwayPolynomial<199, 3> { using ZPZ = aerobus::zpz<199>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; ); // NOLINT template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; }; // NOLINT
03978
                                             template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
03979
                                               template<> struct ConwayPolynomial<199, 6> { using ZPZ = aerobus::zpz<199>; using type =
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<90>, ZPZV<58>, ZPZV<79>, ZPZV<3»; }; // NOLINT
                                             template<> struct ConwayPolynomial<199, 7> { using ZPZ = aerobus::zpz<199>; using type =
 03980
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<39, ZPZV<396%; }; // NOLINT template<> struct ConwayPolynomial<199, 8> { using ZPZ = aerobus::zpz<199>; using type =
 03981
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<160>, ZPZV<23>, ZPZV<159>, ZPZV<3»; }; //
                            NOLINT
03982
                                               template<> struct ConwayPolynomial<199, 9> { using ZPZ = aerobus::zpz<199>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<177>, ZPZV<141>, ZPZV<196»;
                             }; // NOLINT
                            template<> struct ConwayPolynomial<199, 10> { using ZPZ = aerobus::zpz<199>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<158>, ZPZV<31>, ZPZV<54>, ZPZV<54>, ZPZV<9>,
03983
                             ZPZV<3»; }; // NOLINT
                                               template<> struct ConwayPolynomial<199, 11> { using ZPZ = aerobus::zpz<199>; using type
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<1>, ZPZV<196»; }; // NOLINT</pre>
03985
                                             template<> struct ConwayPolynomial<199, 12> { using ZPZ = aerobus::zpz<199>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<33>, ZPZV<192>, ZPZV<197>, ZPZV<138>,
                            ZPZV<69>, ZPZV<57>, ZPZV<151>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<199, 13> { using ZPZ = aerobus::zpz<199>; using type :
03986
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<196»; }; // NOLINT</pre>
03987
                                             template<> struct ConwayPolynomial<199, 17> { using ZPZ = aerobus::zpz<199>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03988
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19*, 
                             NOLINT
03989
                                               template<> struct ConwayPolynomial<211, 1> { using ZPZ = aerobus::zpz<211>; using type =
                            POLYV<ZPZV<1>, ZPZV<209»; }; // NOLINT
                                               template<> struct ConwayPolynomial<211, 2> { using ZPZ = aerobus::zpz<211>; using type =
03990
                             POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; }; // NOLINT
                                            template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<209»; }; // NOLINT
template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
 03992
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<211, 5> { using ZPZ = aerobus::zpz<211>; using type =
03993
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<209»; }; // NOLINT
                                               template<> struct ConwayPolynomial<211, 6> { using ZPZ = aerobus::zpz<211>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<81>, ZPZV<194>, ZPZV<133>, ZPZV<2»; }; // NOLINT
 03995
                                             template<> struct ConwayPolynomial<211, 7> { using ZPZ = aerobus::zpz<211>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<209»; }; // NOLINT template<> struct ConwayPolynomial<211, 8> { using ZPZ = aerobus::zpz<211>; using type =
03996
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<200>, ZPZV<87>, ZPZV<29>, ZPZV<29>; };
                            NOLINT
                                               template<> struct ConwayPolynomial<211, 9> { using ZPZ = aerobus::zpz<211>; using type =
 03997
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<139>, ZPZV<26>, ZPZV<209»;
                             }; // NOLINT
                                               template<> struct ConwayPolynomial<211, 10> { using ZPZ = aerobus::zpz<211>; using type =
03998
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<30>, ZPZV<61>, ZPZV<148>, ZPZV<148>, ZPZV<87>, ZPZV<125>,
                             ZPZV<2»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<211, 11> { using ZPZ = aerobus::zpz<211>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04000
                                            template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<50>, ZPZV<145>, ZPZV<126>, ZPZV<184>,
                             ZPZV<84>, ZPZV<27>, ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<211, 13> { using ZPZ = aerobus::zpz<211>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<209»; }; // NOLINT
                            template<> struct ConwayPolynomial<211, 17> { using ZPZ = aerobus::zpz<211>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
 04002
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<12>, ZPZV<209»; };</pre>
                       template<> struct ConwayPolynomial<211, 19> { using ZPZ = aerobus::zpz<211>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                               7.P.7.V<0>.
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<209»; }; //</pre>
                       NOLINT
04004
                                      template<> struct ConwavPolynomial<223. 1> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYV<ZPZV<1>, ZPZV<220»; }; // NOLINT
                                       template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
 04005
                       POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; }; // NOLINT
 04006
                                     template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
 04007
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<65, ZPZV<163>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<223, 5> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; }; // NOLINT
 04009
                                     template<> struct ConwayPolynomial<223, 6> { using ZPZ = aerobus::zpz<223>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68>, ZPZV<24>, ZPZV<196>, ZPZV<3»; }; // NOLINT
                                      template<> struct ConwayPolynomial<223, 7> { using ZPZ = aerobus::zpz<223>; using type =
04010
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<220»; }; // NOLINT
                                      template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<3»; };
                       NOLINT
04012
                                    template<> struct ConwayPolynomial<223, 9> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<164>, ZPZV<64>, ZPZV<220»;
                       }; // NOLINT
                                        template<> struct ConwayPolynomial<223, 10> { using ZPZ = aerobus::zpz<223>; using type :
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<118>, ZPZV<177>, ZPZV<87>, ZPZV<99>, ZPZV<62>,
                        ZPZV<3»; }; // NOLINT</pre>
04014
                                       template<> struct ConwayPolynomial<223, 11> { using ZPZ = aerobus::zpz<223>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       template<> struct ConwayPolynomial<223, 12> { using ZPZ = aerobus::zpz<223>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<64>, ZPZV<94>, ZPZV<11>, ZPZV<105>, ZPZV<64>,
                       ZPZV<151>, ZPZV<213>, ZPZV<3»; }; // NOLINT</pre>
                       \label{eq:convayPolynomial} $$ \text{template} <> \text{struct ConwayPolynomial} <223, 13> { using ZPZ = aerobus::zpz<223>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
 04016
                       ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<220»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<223, 17> { using ZPZ = aerobus::zpz<223>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; };</pre>
                                                                                                                                                                                                                                                                                                                             // NOLINT
04018
                                    template<> struct ConwayPolynomial<223, 19> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<20»; }; //</pre>
                       NOLINT
                                      template<> struct ConwayPolynomial<227, 1> { using ZPZ = aerobus::zpz<227>; using type =
                       POLYV<ZPZV<1>, ZPZV<225»; }; // NOLINT
 04020
                                    template<> struct ConwayPolynomial<227, 2> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2»; }; // NOLINT
 04021
                                      template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<225»; }; // NOLINT template<> struct ConwayPolynomial<227, 4> { using ZPZ = aerobus::zpz<227>; using type =
04022
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2»; }; // NOLINT
 04023
                                      template<> struct ConwayPolynomial<227, 5> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<225»; }; // NOLINT template<> struct ConwayPolynomial<227, 6> { using ZPZ = aerobus::zpz<227>; using type =
04024
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<135>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<227, 7> { using ZPZ = aerobus::zpz<227>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<225»; };
                                    template<> struct ConwayPolynomial<227, 8> { using ZPZ = aerobus::zpz<227>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<151>, ZPZV<176>, ZPZV<106>, ZPZV<2»; }; //
                       NOLINT
                                    template<> struct ConwayPolynomial<227, 9> { using ZPZ = aerobus::zpz<227>; using type =
04027
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<24>, ZPZV<183>, ZPZV<225»;
                       }; // NOLINT
                                    template<> struct ConwayPolynomial<227, 10> { using ZPZ = aerobus::zpz<227>; using type =
04028
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<129>, ZPZV<12>, ZPZV<12>, ZPZV<93>, ZPZV<77>,
                       ZPZV<2»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type =
04029
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<2>, ZPZV<225»; };</pre>
                                                                                                                 // NOLINT
                                      template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<123>, ZPZV<99>, ZPZV<160>, ZPZV<96>,
                       ZPZV<127>, ZPZV<142>, ZPZV<94>, ZPZV<2»; }; // NOLINT
  template<> struct ConwayPolynomial<227, 13> { using ZPZ = aerobus::zpz<227>; using type :
04031
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25»; }; // NOLINT
    template<> struct ConwayPolynomial<227, 19> { using ZPZ = aerobus::zpz<227>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
04033
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3+, ZPZV<3+, ZPZV<3+, ZPZV<3+</pre>; //
 04034
                                     template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
                     POLYV<ZPZV<1>, ZPZV<223»; }; // NOLINT template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
 04035
                       POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»; }; // NOLINT
```

```
04036
                                   template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
                    POLYY<ZPZY<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; }; // NOLINT template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
04037
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<6»; }; // NOLINT
                                   template<> struct ConwayPolynomial<229, 5> { using ZPZ = aerobus::zpz<229>; using type =
 04038
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223»; }; // NOLINT template<> struct ConwayPolynomial<229, 6> { using ZPZ = aerobus::zpz<229>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<160>, ZPZV<186>, ZPZV<6»; }; // NOLINI
 04040
                                 template<> struct ConwayPolynomial<229, 7> { using ZPZ = aerobus::zpz<229>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<23»; }; // NOLINT
                                 template<> struct ConwayPolynomial<229, 8> { using ZPZ = aerobus::zpz<229>; using type =
04041
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<193>, ZPZV<62>, ZPZV<205>, ZPZV<6*); };
                                   template<> struct ConwayPolynomial<229, 9> { using ZPZ = aerobus::zpz<229>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<117>, ZPZV<117>, ZPZV<50>, ZPZV<223»;
                     }; // NOLINT
                                   template<> struct ConwayPolynomial<229, 10> { using ZPZ = aerobus::zpz<229>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<185>, ZPZV<135>, ZPZV<158>, ZPZV<167>, ZPZV<98>, ZPZV<6»; }; // NOLINT
                                   template<> struct ConwayPolynomial<229, 11> { using ZPZ = aerobus::zpz<229>, using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<223»; }; // NOLINT
                                 template<> struct ConwayPolynomial<229, 12> { using ZPZ = aerobus::zpz<229>; using type =
04045
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<131>, ZPZV<140>, ZPZV<25>, ZPZV<6>, ZPZV<172>,
                     ZPZV<9>, ZPZV<145>, ZPZV<6»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<223»; }; // NOLINT</pre>
04047
                                   template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<223»; }; //</pre>
                     NOLINT
                                   template<> struct ConwayPolynomial<233, 1> { using ZPZ = aerobus::zpz<233>; using type =
04049
                    POLYV<ZPZV<1>, ZPZV<230»; }; // NOLINT
                                   template<> struct ConwayPolynomial<233, 2> { using ZPZ = aerobus::zpz<233>; using type =
                    POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
                                 template<> struct ConwayPolynomial<233, 3> { using ZPZ = aerobus::zpz<233>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
04052
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<233, 5> { using ZPZ = aerobus::zpz<233>; using type =
 04053
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<230»; }; // NOLINT
 04054
                                   template<> struct ConwayPolynomial<233, 6> { using ZPZ = aerobus::zpz<233>; using type =
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<215>, ZPZV<32>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<233, 7> { using ZPZ = aerobus::zpz<233>; using type =
04055
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230»; }; // NOLINT
                                 template<> struct ConwayPolynomial<233, 8> { using ZPZ = aerobus::zpz<233>; using type =
04056
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<202>, ZPZV<135>, ZPZV<181>, ZPZV<3»; }; //
04057
                                 template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<146>, ZPZV<230»;
                     }; // NOLINT
04058
                                   template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<28>, ZPZV<71>, ZPZV<102>, ZPZV<3>, ZPZV<48>,
                     ZPZV<3»: }: // NOLINT
                                   template<> struct ConwayPolynomial<233, 11> { using ZPZ = aerobus::zpz<233>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<5>, ZPZV<230»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<233, 12> { using ZPZ = aerobus::zpz<233>; using type :
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<21>, ZPZV<114>, ZPZV<31>, ZPZV<19>,
                     ZPZV<216>, ZPZV<20>, ZPZV<3»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<233, 13> { using ZPZ = aerobus::zpz<233>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                    ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<230»; }; // NOLINT
template<> struct ConwayPolynomial<233, 17> { using ZPZ = aerobus::zpz<233>; using type =
04062
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230»; };</pre>
                                                                                                                                                                                                                                                                                                // NOLINT
                                   template<> struct ConwayPolynomial<233, 19> { using ZPZ = aerobus::zpz<233>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<25>, ZPZV<230»; }; //</pre>
                     NOLINT
04064
                                   template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
                     POLYV<ZPZV<1>, ZPZV<232»; }; // NOLINT
                                  template<> struct ConwayPolynomial<239, 2> { using ZPZ = aerobus::zpz<239>; using type =
                    POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7»; }; // NOLINT
 04066
                                  template<> struct ConwayPolynomial<239, 3> { using ZPZ = aerobus::zpz<239>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<232»; }; // NOLINT
template<> struct ConwayPolynomial<239, 4> { using ZPZ = aerobus::zpz<239>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7»; }; // NOLINT
 04067
                                   template<> struct ConwayPolynomial<239, 5> { using ZPZ = aerobus::zpz<239>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; }; // NOLINT
 04069
                                template<> struct ConwayPolynomial<239, 6> { using ZPZ = aerobus::zpz<239>; using type =
                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<237>, ZPZV<60>, ZPZV<200>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<239, 7> { using ZPZ = aerobus::zpz<239>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<17>, ZPZV<232»; };
                                template<> struct ConwayPolynomial<239, 8> { using ZPZ = aerobus::zpz<239>; using type
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<202>, ZPZV<54>, ZPZV<7»; }; //
                  NOLINT
04072
                              template<> struct ConwayPolynomial<239, 9> { using ZPZ = aerobus::zpz<239>; using type =
                  POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<88>, ZPZV<232»; };
                                template<> struct ConwayPolynomial<239, 10> { using ZPZ = aerobus::zpz<239>; using type =
04073
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<68>, ZPZV<226>, ZPZV<127>, ZPZV<108>, ZPZV<7»; }; // NOLINT
                  template<> struct ConwayPolynomial<239, 11> { using ZPZ = aerobus::zpz<239>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04074
                   ZPZV<8>, ZPZV<232»; };</pre>
                                                                                               // NOLINT
                                template<> struct ConwayPolynomial<239, 12> { using ZPZ = aerobus::zpz<239>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<235>, ZPZV<14>, ZPZV<113>, ZPZV<162>,
                   ZPZV<101>, ZPZV<81>, ZPZV<216>, ZPZV<7»; }; // NOLINT
  template<> struct ConwayPolynomial<239, 13> { using ZPZ = aerobus::zpz<239>; using type =
04076
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                template<> struct ConwayPolynomial<239, 17> { using ZPZ = aerobus::zpz<239>; using type
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                                                                                                                                                                                                                                                         // NOLINT
                   template<> struct ConwayPolynomial<239, 19> { using ZPZ = aerobus::zpz<239>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2+>, ZPZ
04078
                   NOLINT
                               template<> struct ConwayPolynomial<241, 1> { using ZPZ = aerobus::zpz<241>; using type =
04079
                  POLYV<ZPZV<1>, ZPZV<234»; }; // NOLINT
04080
                                template<> struct ConwayPolynomial<241, 2> { using ZPZ = aerobus::zpz<241>; using type =
                  POLYV<ZPZV<1>, ZPZV<238>, ZPZV<7»; }; // NOLINT
                              template<> struct ConwayPolynomial<241, 3> { using ZPZ = aerobus::zpz<241>; using type =
04081
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<234»; }; // NOLINT
                               template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
04082
                   \verb"POLYV<ZPZV<1>, \ \verb"ZPZV<0>, \ \verb"ZPZV<14>, \ \verb"ZPZV<152>, \ \verb"ZPZV<7"; \ \verb"}; \ \ // \ \verb"NOLINT" 
04083
                                template<> struct ConwayPolynomial<241, 5> { using ZPZ = aerobus::zpz<241>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<234»; }; // NOLINT
                               template<> struct ConwayPolynomial<241, 6> { using ZPZ = aerobus::zpz<241>; using type =
04084
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<6>, ZPZV<5>, ZPZV<7»; }; // NOLINT
04085
                                template<> struct ConwayPolynomial<241, 7> { using ZPZ = aerobus::2pz<241>; using type
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<234»; }; // NOLINT
04086
                              template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<173>, ZPZV<212>, ZPZV<153>, ZPZV<153>, ZPZV<7»; }; //
                   NOLINT
04087
                              template<> struct ConwayPolynomial<241, 9> { using ZPZ = aerobus::zpz<241>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<236>, ZPZV<125>, ZPZV<234»;
                   }; // NOLINT
04088
                              template<> struct ConwayPolynomial<241, 10> { using ZPZ = aerobus::zpz<241>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<27>, ZPZV<145>, ZPZV<208>, ZPZV<55>,
                   ZPZV<7»: }: // NOLINT
                                template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type =
04089
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<3>, ZPZV<234»; }; // NOLINT</pre>
04090
                              template<> struct ConwayPolynomial<241, 12> { using ZPZ = aerobus::zpz<241>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<10>, ZPZV<109>, ZPZV<168>, ZPZV<22>, ZPZV<197>, ZPZV<17>, ZPZV<7»; }; // NOLINT
04091
                                template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type :
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<241, 17> { using ZPZ = aerobus::zpz<241>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<234»; }; // NOLINT
   template<> struct ConwayPolynomial<241, 19> { using ZPZ = aerobus::zpz<241>; using type :
04093
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234»; }; //</pre>
                   NOLINT
04094
                              template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
                  POLYV<ZPZV<1>, ZPZV<245»; }; // NOLINT
                              template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
04095
                  POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; }; // NOLINT
                                template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
04096
                  POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<245»; }; // NOLINT template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type =
04097
                 POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<200>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<251, 5> { using ZPZ = aerobus::zpz<251; using type =
04098
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT template<> struct ConwayPolynomial<251, 6> { using ZPZ = aerobus::zpz<251>; using type =
                  POLYV<2PZV<1>, 2PZV<0>, ZPZV<1>, ZPZV<24>, ZPZV<25>, ZPZV<179>, ZPZV<6»; }; // NOLINT
04100
                              template<> struct ConwayPolynomial<251, 7> { using ZPZ = aerobus::zpz<251>; using type =
                 Template<> struct ConwayFolynomial<21f, /> { using ZFZ = aerobus:.2pz<21f, using Cype = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>; // NOLINT template<> struct ConwayFolynomial<251, 8> { using ZPZ = aerobus::2pz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<142>, ZPZV<215>, ZPZV<173>, ZPZV<6»; }; //
04101
                              template<> struct ConwayPolynomial<251, 9> { using ZPZ = aerobus::zpz<251>; using type
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<106>, ZPZV<245»;
                   }; // NOLINT
                  template<> struct ConwayPolynomial<251, 10> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<138>, ZPZV<110>, ZPZV<45>, ZPZV<34>,
04103
```

```
ZPZV<149>, ZPZV<6»; };</pre>
                                               template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<26>, ZPZV<245»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<53>, ZPZV<20>, ZPZV<20>, ZPZV<15>, ZPZV<201>, ZPZV<202>, ZPZV<20>, ZP
04105
                                               template<> struct ConwayPolynomial<251, 13> { using ZPZ = aerobus::zpz<251>; using type :
04106
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<245»; }; // NOLINT
  template<> struct ConwayPolynomial<251, 17> { using ZPZ = aerobus::zpz<251>; using type =
04107
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<245»; }; // NOLINT
template<> struct ConwayPolynomial<251, 19> { using ZPZ = aerobus::zpz<251>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<245»; }; //</pre>
                             NOLINT
                                              template<> struct ConwayPolynomial<257, 1> { using ZPZ = aerobus::zpz<257>; using type =
04109
                            POLYV<ZPZV<1>, ZPZV<254»; }; // NOLINT
                                              template<> struct ConwayPolynomial<257, 2> { using ZPZ = aerobus::zpz<257>; using type =
                            POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3»; }; // NOLINT
 04111
                                              template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<254»; }; // NOLINT template<> struct ConwayPolynomial<257, 4> { using ZPZ = aerobus::zpz<257>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<187>, ZPZV<3»; }; // NOLINT
 04112
                                               template<> struct ConwayPolynomial<257, 5> { using ZPZ = aerobus::zpz<257>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<254»; }; // NOLINT
 04114
                                              template<> struct ConwayPolynomial<257, 6> { using ZPZ = aerobus::zpz<257>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<62>, ZPZV<18>, ZPZV<138>, ZPZV<3»; }; // NOLINT
                          template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<254»; }; // NOLINT
04115
 04116
                                               template<> struct ConwayPolynomial<257, 8> { using ZPZ = aerobus::zpz<257>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<179>, ZPZV<140>, ZPZV<162>, ZPZV<3»; }; //
                           template<> struct ConwayPolynomial<257, 9> { using ZPZ = aerobus::zpz<257>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<201>, ZPZV<50>, ZPZV<254»;</pre>
04117
                            }; // NOLINT
                                                 template<> struct ConwayPolynomial<257, 10> { using ZPZ = aerobus::zpz<257>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<225>, ZPZV<26>, ZPZV<20>,
                             ZPZV<3»; }; // NOLINT</pre>
04119
                                            template<> struct ConwayPolynomial<257, 11> { using ZPZ = aerobus::zpz<257>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<40>, ZPZV<254»; }; // NOLINT
    template<> struct ConwayPolynomial<257, 12> { using ZPZ = aerobus::zpz<257>; using type =
04120
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<23>, ZPZV<225>, ZPZV<215>, ZPZV<2173>, ZPZV<249>, ZPZV<148>, ZPZV<20>, ZPZV<3»; }; // NOLINT
04121
                                            template<> struct ConwayPolynomial<257, 13> { using ZPZ = aerobus::zpz<257>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              template<> struct ConwayPolynomial<257, 17> { using ZPZ = aerobus::zpz<257>; using type =
04122
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<8>, ZPZV<254»; };</pre>
04123
                                            template<> struct ConwayPolynomial<257, 19> { using ZPZ = aerobus::zpz<257>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<254»; }; //</pre>
                             NOLINT
                                               template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
                            POLYV<ZPZV<1>, ZPZV<258»; }; // NOLINT
                                               template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
                            POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; }; // NOLINT
                                             template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
04126
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; }; // NOLINT template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; }; // NOLINT
04128
                                            template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<258»; }; // NOLINT
 04129
                                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<222>, ZPZV<250>, ZPZV<225>, ZPZV<25>, ZPZV<263>; // NOLINT template<> struct ConwayPolynomial<263, 7> { using ZPZ = aerobus::zpz<263>; using type
04130
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<258»; }; //
 04131
                                              template<> struct ConwayPolynomial<263, 8> { using ZPZ = aerobus::zpz<263>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<227>, ZPZV<170>, ZPZV<7>, ZPZV<5»; };
                            NOLINT
04132
                                              template<> struct ConwayPolynomial<263, 9> { using ZPZ = aerobus::zpz<263>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<261>, ZPZV<261>, ZPZV<29>, ZPZV<258»;
                                            template<> struct ConwayPolynomial<263, 10> { using ZPZ = aerobus::zpz<263>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<245>, ZPZV<231>, ZPZV<198>, ZPZV<145>, ZPZV<119>, ZPZV<5»; }; // NOLINT
                                            template<> struct ConwayPolynomial<263, 11> { using ZPZ = aerobus::zpz<263>; using type =
04134
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<2>, ZPZV<258»; };</pre>
                                                                                                                                             // NOLINT
                           template<> struct ConwayPolynomial<263, 12> { using ZPZ = aerobus::zpz<263>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<174>, ZPZV<162>, ZPZV<162>, ZPZV<252>,
                            ZPZV<47>, ZPZV<45>, ZPZV<1800, ZPZV<5»; }; // NOLINT
   template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
POLYV<ZPZV<1>, ZPZV<267»; }; // NOLINT</pre>
```

```
04137
                          template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
               POLYV<ZPZV<1>, ZPZV<268>, ZPZV<2»; }; // NOLINT
                        template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<267»; }; // NOLINT
template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2»; }; // NOLINT
04139
                         template<> struct ConwayPolynomial<269, 5> { using ZPZ = aerobus::zpz<269>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<267»; }; // NOLINT
04141
                        template<> struct ConwayPolynomial<269, 6> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<10>, ZPZV<206>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<269, 7> { using ZPZ = aerobus::zpz<269>; using type =
04142
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<267»; }; // NOLINT
                         template<> struct ConwayPolynomial<269, 8> { using ZPZ = aerobus::zpz<269>; using type =
04143
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<220>, ZPZV<131>, ZPZV<232>, ZPZV<23x; }; //
               NOLINT
                        template<> struct ConwayPolynomial<269, 9> { using ZPZ = aerobus::zpz<269>; using type =
04144
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<214>, ZPZV<267>, ZPZV<267»;
               }; // NOLINT
                         template<> struct ConwayPolynomial<269, 10> { using ZPZ = aerobus::zpz<269>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<264>, ZPZV<243>, ZPZV<186>, ZPZV<61>,
               ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
04146
                         template<> struct ConwayPolynomial<269, 11> { using ZPZ = aerobus::zpz<269>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<20>, ZPZV<267»; }; // NOLINT
  template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type =
04147
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<165>, ZPZV<63>, ZPZV<215>,
               04148
                        template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<265»; }; // NOLINT
                        template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
04149
              POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; }; // NOLINT
04150
                         template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
               POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<265»; }; // NOLINT template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type =
04151
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<271, 5> { using ZPZ = aerobus::zpz<271>; using type =
04152
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT template<> struct ConwayPolynomial<271, 6> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<207>, ZPZV<207>, ZPZV<81>, ZPZV<6»; }; // NOLINT
                        template<> struct ConwayPolynomial<271, 7> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<22>, ZPZV<265»; };
                        template<> struct ConwayPolynomial<271, 8> { using ZPZ = aerobus::zpz<271>; using type =
04155
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>; };
                         template<> struct ConwayPolynomial<271, 9> { using ZPZ = aerobus::zpz<271>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<266>, ZPZV<186>, ZPZV<265»;
               }; // NOLINT
               template<> struct ConwayPolynomial<271, 10> { using ZPZ = aerobus::zpz<271>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<10>, ZPZV<256>, ZPZV<74>,
04157
               ZPZV<126>, ZPZV<6»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type
04158
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<10>, ZPZV<265»; }; // NOLINT</pre>
               \label{eq:convergence} template<> struct ConwayPolynomial<271, 12> \{ using ZPZ = aerobus::zpz<271>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162>, ZPZV<210>, ZPZV<216>, ZPZV<210>, ZPZV<205>, ZPZV<205>, ZPZV<210>, ZPZV<210
04159
               ZPZV<237>, ZPZV<256>, ZPZV<130>, ZPZV<6»; }; // NOLINT
  template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<272»; }; // NOLINT
                         template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; }; // NOLINT
                        template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
04162
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; }; // NOLINT template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
04163
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<5»; }; // NOLINT
04164
                        template<> struct ConwayPolynomial<277, 5> { using ZPZ = aerobus::zpz<277>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<272»; }; // NOLINT
              template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<18>, ZPZV<5»; }; // NOLINT
04165
                        template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type
04166
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<272»; }; //
                         template<> struct ConwayPolynomial<277, 8> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<159>, ZPZV<176>, ZPZV<5»; }; //
               NOLINT
                         template<> struct ConwayPolynomial<277, 9> { using ZPZ = aerobus::zpz<277>; using type =
04168
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177>, ZPZV<110>, ZPZV<272»;
                        template<> struct ConwayPolynomial<277, 10> { using ZPZ = aerobus::zpz<277>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<253>, ZPZV<237>, ZPZV<241>, ZPZV<260>, ZPZV<5»; }; // NOLINT
04170
                        template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<5>, ZPZV<272»; };</pre>
                                                                            // NOLINT
               template<> struct ConwayPolynomial<277, 12> { using ZPZ = aerobus::zpz<277>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<183>, ZPZV<218>, ZPZV<240>, ZPZV<40>,
               <code>ZPZV<180>, ZPZV<115>, ZPZV<202>, ZPZV<5»; }; // NOLINT</code>
               template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
POLYV<ZPZV<1>, ZPZV<278»; }; // NOLINT</pre>
04172
```

```
template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
              POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; }; // NOLINT
04174
                       template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<278»; }; // NOLINT
template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3»; }; // NOLINT
04175
                         template<> struct ConwayPolynomial<281, 5> { using ZPZ = aerobus::zpz<281>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<278»; }; // NOLINT
04177
                        template<> struct ConwayPolynomial<281, 6> { using ZPZ = aerobus::zpz<281>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<15), ZPZV<151>, ZPZV<13>, ZPZV<27>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<281, 7> { using ZPZ = aerobus::zpz<281>; using type =
04178
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<19>, ZPZV<278»; }; // NOLINT
                         template<> struct ConwayPolynomial<281, 8> { using ZPZ = aerobus::zpz<281>; using type =
04179
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195>, ZPZV<279>, ZPZV<140>, ZPZV<3»; }; //
               NOLINT
                        template<> struct ConwayPolynomial<281, 9> { using ZPZ = aerobus::zpz<281>; using type =
04180
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<148>, ZPZV<70>, ZPZV<278»;
               }; // NOLINT
                         template<> struct ConwayPolynomial<281, 10> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<145>, ZPZV<138>,
               ZPZV<191>, ZPZV<3»; }; // NOLINT</pre>
04182
                         template<> struct ConwayPolynomial<281, 11> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
              ZPZV<36>, ZPZV<278»; }; // NOLINT
  template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type =
04183
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<202>, ZPZV<68>, ZPZV<103>, ZPZV<116>,
               ZPZV<58>, ZPZV<28>, ZPZV<191>, ZPZV<3»; }; // NOLINT</pre>
04184
                        template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
              POLYV<ZPZV<1>, ZPZV<280»; }; // NOLINT
                       template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
04185
              POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3»; }; // NOLINT
04186
                         template<> struct ConwayPolynomial<283, 3> { using ZPZ = aerobus::zpz<283>; using type =
              POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<280»; }; // NOLINT template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type =
04187
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
04188
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
                         template<> struct ConwayPolynomial<283, 6> { using ZPZ = aerobus::zpz<283>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<68>, ZPZV<73>, ZPZV<3»; };
                       template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<280»; }; // NOLINT
                        template<> struct ConwayPolynomial<283, 8> { using ZPZ = aerobus::zpz<283>; using type =
04191
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<179>, ZPZV<32>, ZPZV<232>, ZPZV<23»; }; //
                         template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               }; // NOLINT
                         template<> struct ConwayPolynomial<283, 10> { using ZPZ = aerobus::zpz<283>; using type =
04193
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<271>, ZPZV<185>, ZPZV<68>, ZPZV<100>,
               ZPZV<219>, ZPZV<3»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<283, 11> { using ZPZ = aerobus::zpz<283>; using type
04194
               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<4>, ZPZV<280»; }; // NOLINT</pre>
              template<> struct ConwayPolynomial<283, 12> { using ZPZ = aerobus::zpz<283>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<8>, ZPZV<8>, ZPZV<96>, ZPZV<29>, ZPZV<49>,
04195
               ZPZV<14>, ZPZV<56>, ZPZV<3»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<293, 1> { using ZPZ = aerobus::zpz<293>; using type =
              POLYV<ZPZV<1>, ZPZV<291»; }; // NOLINT
                         template<> struct ConwayPolynomial<293, 2> { using ZPZ = aerobus::zpz<293>; using type =
04197
              POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; }; // NOLINT
                        template<> struct ConwayPolynomial<293, 3> { using ZPZ = aerobus::zpz<293>; using type =
04198
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT template<> struct ConwayPolynomial<293, 4> { using ZPZ = aerobus::zpz<293>; using type =
04199
              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
04200
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
04201
                         template<> struct ConwayPolynomial<293, 6> { using ZPZ = aerobus::zpz<293>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<260>, ZPZV<260>, ZPZV<28>; // NOLINT template<> struct ConwayPolynomial<293, 7> { using ZPZ = aerobus::zpz<293>; using type
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<291»; }; //
                        template<> struct ConwayPolynomial<293, 8> { using ZPZ = aerobus::zpz<293>; using type =
04203
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<239; }; //
               NOLINT
04204
                        template<> struct ConwayPolynomial<293, 9> { using ZPZ = aerobus::zpz<293>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<208>, ZPZV<190>, ZPZV<291»;
04205
                        template<> struct ConwayPolynomial<293, 10> { using ZPZ = aerobus::zpz<293>; using type :
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<28>, ZPZV<26>, ZPZV<46>, ZPZV<24>,
               ZPZV<2»; }; // NOLINT</pre>
04206
                       template<> struct ConwayPolynomial<293, 11> { using ZPZ = aerobus::zpz<293>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<3>, ZPZV<291»; };</pre>
                                                                           // NOLINT
              template<> struct ConwayPolynomial<293, 12> { using ZPZ = aerobus::zpz<293>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<159>, ZPZV<210>, ZPZV<125>, ZPZV<212>,
               <code>ZPZV<167></code>, <code>ZPZV<144></code>, <code>ZPZV<157></code>, <code>ZPZV<2»</code>; <code>}</code>; <code>// NOLINT</code>
              template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
POLYV<ZPZV<1>, ZPZV<302»; }; // NOLINT</pre>
04208
```

```
04209
               template<> struct ConwayPolynomial<307, 2> { using ZPZ = aerobus::zpz<307>; using type =
         POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<302»; }; // NOLINT
template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<307, 5> { using ZPZ = aerobus::zpz<307>; using type =
04211
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<302»; }; // NOLINT
04213
              template<> struct ConwayPolynomial<307, 6> { using ZPZ = aerobus::zpz<307>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<213>, ZPZV<172>, ZPZV<61>, ZPZV<5»; ); // NOLINT template<> struct ConwayPolynomial<307, 7> { using ZPZ = aerobus::zpz<307>; using type =
04214
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<302»; }; // NOLINT
               template<> struct ConwayPolynomial<307, 8> { using ZPZ = aerobus::zpz<307>; using type =
04215
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<283>, ZPZV<232>, ZPZV<131>, ZPZV<5»; }; //
         NOLINT
04216
              template<> struct ConwayPolynomial<307, 9> { using ZPZ = aerobus::zpz<307>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<70>, ZPZV<302»;
         }; // NOLINT
               template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
         POLYV<ZPZV<1>, ZPZV<294»; }; // NOLINT
               template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
         POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; }; // NOLINT
              template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
04219
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; }; // NOLINT template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
04220
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; };
                                                                                              // NOLINT
               template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
04221
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294»; }; // NOLINT
04222
               template<> struct ConwayPolynomial<311, 6> { using ZPZ = aerobus::zpz<311>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<167>, ZPZV<152>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<311, 7> { using ZPZ = aerobus::zpz<311>; using type
04223
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<294»; };
               template<> struct ConwayPolynomial<311, 8> { using ZPZ = aerobus::zpz<311>; using type
04224
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<162>, ZPZV<118>, ZPZV<2>, ZPZV<27»; }; //
         NOLINT
04225
               template<> struct ConwayPolynomial<311, 9> { using ZPZ = aerobus::zpz<311>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<287>, ZPZV<287>, ZPZV<294»;
04226
               template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
         POLYV<ZPZV<1>, ZPZV<303»; }; // NOLINT
04227
              template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
         POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
04228
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; }; // NOLINT
               template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<239>, ZPZV<10»; }; // NOLINT
04230
              template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<303»; }; // NOLINT
               template<> struct ConwayPolynomial<313, 6> { using ZPZ = aerobus::zpz<313>; using type =
04231
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<196>, ZPZV<213>, ZPZV<253>, ZPZV<10»; }; // NOLINT
04232
               template<> struct ConwayPolynomial<313,
                                                                          7> { using ZPZ = aerobus::zpz<313>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
              template<> struct ConwayPolynomial<313, 8> { using ZPZ = aerobus::zpz<313>; using type
04233
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<306>, ZPZV<99>, ZPZV<106>, ZPZV<10»; }; //
         NOLINT
         template<> struct ConwayPolynomial<313, 9> { using ZPZ = aerobus::zpz<313>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<303»;</pre>
04234
         }; // NOLINT
               template<> struct ConwayPolynomial<317, 1> { using ZPZ = aerobus::zpz<317>; using type =
04235
         POLYV<ZPZV<1>, ZPZV<315»; }; // NOLINT
              template<> struct ConwayPolynomial<317, 2> { using ZPZ = aerobus::zpz<317>; using type =
04236
         POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; }; // NOLINT
04237
               template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<7>, ZPZY<315»; }; // NOLINT template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
04238
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<317, 5> { using ZPZ = aerobus::zpz<317>; using type =
04239
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZV<0>, ZPZV<2>, ZPZV<25; }; // NOLINT template<> struct ConwayPolynomial<317, 6> { using ZPZ = aerobus::zpz<317>; using type =
04240
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<195>, ZPZV<156>, ZPZV<4>, ZPZV<4»; }; // NOLINT
               template<> struct ConwayPolynomial<317, 7> { using ZPZ = aerobus::zpz<317>; using type =
04241
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<315»; }; // NOLINT
04242
              template<> struct ConwayPolynomial<317, 8> { using ZPZ = aerobus::zpz<317>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<207>, ZPZV<85>, ZPZV<31>, ZPZV<2»; };
         NOLINT
               template<> struct ConwayPolynomial<317, 9> { using ZPZ = aerobus::zpz<317>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<284>, ZPZV<296>, ZPZV<315»;
         }; // NOLINT
04244
               template<> struct ConwayPolynomial<331, 1> { using ZPZ = aerobus::zpz<331>; using type =
         POLYV<ZPZV<1>, ZPZV<328»; }; // NOLINT
               template<> struct ConwayPolynomial<331, 2> { using ZPZ = aerobus::zpz<331>; using type =
04245
         POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<328»; };
                                                                                // NOLINT
              template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
04247
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<290>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<331, 5> { using ZPZ = aerobus::zpz<331>; using type =
04248
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<328»; };
            template<> struct ConwayPolynomial<331, 6> { using ZPZ = aerobus::zpz<331>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<205>, ZPZV<159>, ZPZV<3»; }; // NOLINT
04250
           template<> struct ConwayPolynomial<331, 7> { using ZPZ = aerobus::zpz<331>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<328»; }; // NOLINT template<> struct ConwayPolynomial<331, 8> { using ZPZ = aerobus::zpz<331>; using type =
04251
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<249>, ZPZV<308>, ZPZV<78>, ZPZV<3»; };
04252
           template<> struct ConwayPolynomial<331, 9> { using ZPZ = aerobus::zpz<331>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<194>, ZPZV<210>, ZPZV<328»;
       }; // NOLINT
04253
            template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
       POLYV<ZPZV<1>, ZPZV<327»; }; // NOLINT
            template<> struct ConwayPolynomial<337, 2> { using ZPZ = aerobus::zpz<337>; using type =
       POLYV<ZPZV<1>, ZPZV<332>, ZPZV<10»; }; // NOLINT
       template<> struct ConwayPolynomial<337, 3> { using ZPZ = aerobus::zpz<337>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
04255
04256
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10»; }; // NOLINT
            template<> struct ConwayPolynomial<337, 5> { using ZPZ = aerobus::zpz<337>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT
04258
            template<> struct ConwayPolynomial<337, 6> { using ZPZ = aerobus::zpz<337>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<216>, ZPZV<12>, ZPZV<10>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<337, 7> { using ZPZ = aerobus::zpz<337>; using type
04259
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<327»; }; // NOLINT
            template<> struct ConwayPolynomial<337, 8> { using ZPZ = aerobus::zpz<337>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<331>, ZPZV<246>, ZPZV<251>, ZPZV<10»; }; //
       NOLINT
       template<> struct ConwayPolynomial<337, 9> { using ZPZ = aerobus::zpz<337>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<148>, ZPZV<98>, ZPZV<327»;</pre>
04261
       }; // NOLINT
04262
            template<> struct ConwayPolynomial<347, 1> { using ZPZ = aerobus::zpz<347>; using type =
       POLYV<ZPZV<1>, ZPZV<345»; }; // NOLINT
           template<> struct ConwayPolynomial<347, 2> { using ZPZ = aerobus::zpz<347>; using type =
04263
       POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
04264
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<9>, ZPZY<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; };
                                                                             // NOLINT
           template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<345»; }; // NOLINT
04267
            template<> struct ConwayPolynomial<347, 6> { using ZPZ = aerobus::zpz<347>; using type =
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<343>, ZPZV<26>, ZPZV<56>, ZPZV<2»; ); // NOLINT template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type
04268
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<345»; }; // NOLINT
04269
            template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<187>, ZPZV<213>, ZPZV<117>, ZPZV<2»; }; //
       NOLINT
04270
           template<> struct ConwayPolynomial<347, 9> { using ZPZ = aerobus::zpz<347>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<255>, ZPZV<345»;
       }; // NOLINT
  template<> struct ConwayPolynomial<349, 1> { using ZPZ = aerobus::zpz<349>; using type =
       POLYV<ZPZV<1>, ZPZV<347»; }; // NOLINT template<> struct ConwayPolynomial<349, 2> { using ZPZ = aerobus::zpz<349>; using type =
04272
       POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
04273
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; }; // NOLINT
           template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
04274
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<279>, ZPZV<2%; }; // NOLINT template<> struct ConwayPolynomial<349, 5> { using ZPZ = aerobus::zpz<349>; using type =
04275
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<347»; }; // NOLINT template<> struct ConwayPolynomial<349, 6> { using ZPZ = aerobus::zpz<349>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<135>, ZPZV<177>, ZPZV<316>, ZPZV<2»; }; // NOLINT
04276
            template<> struct ConwayPolynomial<349, 7> { using ZPZ = aerobus::zpz<349>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<347»; };
04278
           template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<328>, ZPZV<268>, ZPZV<2*; }; //
       NOLINT
           template<> struct ConwayPolynomial<349, 9> { using ZPZ = aerobus::zpz<349>; using type =
04279
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<290>, ZPZV<130>, ZPZV<347»;
       }; // NOLINT
04280
           template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<ZPZV<1>, ZPZV<350»; }; // NOLINT
            template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
04281
       POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type =
04283
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
04284
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<350»; }; // NOLINT
            template<> struct ConwayPolynomial<353, 6> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<215>, ZPZV<226>, ZPZV<295>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<353, 7> { using ZPZ = aerobus::zpz<353>; using type =
04286
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<16>, ZPZV<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 8> { using ZPZ = aerobus::zpz<353>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<26>, ZPZV<37>, ZPZV<37>, ZPZV<3»; };
04287
```

```
NOLINT
           template<> struct ConwayPolynomial<353, 9> { using ZPZ = aerobus::zpz<353>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<319>, ZPZV<49>, ZPZV<350»;
04288
           }; // NOLINT
04289
                   template<> struct ConwayPolynomial<359, 1> { using ZPZ = aerobus::zpz<359>; using type =
           POLYV<ZPZV<1>, ZPZV<352»; }; // NOLINT
                   template<> struct ConwayPolynomial<359, 2> { using ZPZ = aerobus::zpz<359>; using type =
           POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7»; }; // NOLINT
                  template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
04291
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<352»; ); // NOLINT template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
04292
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<229>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<359, 5> { using ZPZ = aerobus::zpz<359>; using type =
04293
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; }; // NOLINT
04294
                  template<> struct ConwayPolynomial<359, 6> { using ZPZ = aerobus::zpz<359>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<309>, ZPZV<327>, ZPZV<327>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<359, 7> { using ZPZ = aerobus::zpz<359>; using type =
04295
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; }; // NOLINT
                  template<> struct ConwayPolynomial<359, 8> { using ZPZ = aerobus::zpz<359>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<301>, ZPZV<143>, ZPZV<271>, ZPZV<7»; }; //
04297
                  template<> struct ConwayPolynomial<359, 9> { using ZPZ = aerobus::zpz<359>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<356>, ZPZV<165>, ZPZV<352»;
           }; // NOLINT
04298
                   template<> struct ConwayPolynomial<367, 1> { using ZPZ = aerobus::zpz<367>; using type =
           POLYV<ZPZV<1>, ZPZV<361»; }; // NOLINT
                  template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
04299
           POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6»; }; // NOLINT
04300
                  template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<361»; }; // NOLINT template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6»; }; // NOLINT
04301
                  template<> struct ConwayPolynomial<367, 5> { using ZPZ = aerobus::zpz<367>; using type =
04302
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<361»; }; // NOLINT
04303
                  template<> struct ConwayPolynomial<367, 6> { using ZPZ = aerobus::zpz<367>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<222>, ZPZV<321>, ZPZV<324>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type
04304
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<13>, ZPZV<361»; }; // NOLINT
                  template<> struct ConwayPolynomial<367, 8> { using ZPZ = aerobus::zpz<367>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<335>, ZPZV<282>, ZPZV<50>, ZPZV<6»; };
           template<> struct ConwayPolynomial<367, 9> { using ZPZ = aerobus::zpz<367>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<213>, ZPZV<213>, ZPZV<268>, ZPZV<361»;</pre>
04306
           }; // NOLINT
                  template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
           POLYV<ZPZV<1>, ZPZV<371»; }; // NOLINT
04308
                  template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
          POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2»; }; // NOLINT
04309
                  template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<5\, ZPZY<371»; }; // NOLINT template<> struct ConwayPolynomial<373, 4> { using ZPZ = aerobus::zpz<373>; using type =
04310
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304>, ZPZV<2»; }; // NOLINT
04311
                   template<> struct ConwayPolynomial<373, 5> { using ZPZ = aerobus::zpz<373>; using type =
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<371»; }; // NOLINT template<> struct ConwayPolynomial<373, 6> { using ZPZ = aerobus::zpz<373>; using type =
04312
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<126>, ZPZV<108>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<371»; };
                  template<> struct ConwayPolynomial<373, 8> { using ZPZ = aerobus::zpz<373>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<203>, ZPZV<219>, ZPZV<66>, ZPZV<2*; }; //
           NOLINT
           template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<370>, ZPZ
04315
           }; // NOLINT
04316
                  template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
           POLYV<ZPZV<1>, ZPZV<377»; }; // NOLINT
04317
                   template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
           POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<379, 3> { using ZPZ = aerobus::zpz<379>; using type =
04318
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377»; }; // NOLINT template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
04319
           template<> struct ConwayPolynomial<379, 5> { using ZPZ = aerobus::zpz<379>; using type =
04320
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<377»; }; // NOLINT
                  template<> struct ConwayPolynomial<379, 6> { using ZPZ = aerobus::zpz<379>; using type =
04321
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<374>, ZPZV<364>, ZPZV<246>, ZPZV<2»; }; // NOLINI
                  template<> struct ConwayPolynomial<379, 7> { using ZPZ = aerobus::zpz<379>; using type
04322
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<377%; }; // NOLINT template<> struct ConwayPolynomial<379, 8> { using ZPZ = aerobus::zpz<379>; using type =
04323
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<210>, ZPZV<194>, ZPZV<173>, ZPZV<2*; }; //
           NOLINT
04324
                  template<> struct ConwayPolynomial<379, 9> { using ZPZ = aerobus::zpz<379>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<362>, ZPZV<369>, ZPZV<377»;
           }; // NOLINT
04325
                  \texttt{template<> struct ConwayPolynomial<383, 1> \{ using ZPZ = aerobus::zpz<383>; using type = 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 
           POLYV<ZPZV<1>, ZPZV<378»; }; // NOLINT
                  template<> struct ConwayPolynomial<383, 2> { using ZPZ = aerobus::zpz<383>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; };
               template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
04328
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<383, 5> { using ZPZ = aerobus::zpz<383>; using type =
04329
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
               template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
04330
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<8>, ZPZV<158>, ZPZV<5»; }; // NOLINT
04331
              template<> struct ConwayPolynomial<383, 7> { using ZPZ = aerobus::zpz<383>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04332
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<281>, ZPZV<332>, ZPZV<296>, ZPZV<5»; }; //
04333
              template<> struct ConwayPolynomial<383, 9> { using ZPZ = aerobus::zpz<383>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137>, ZPZV<76>, ZPZV<378»;
         }; // NOLINT
               template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
04334
         POLYV<ZPZV<1>, ZPZV<387»; }; // NOLINT
               template<> struct ConwayPolynomial<389, 2> { using ZPZ = aerobus::zpz<389>; using type =
         POLYV<ZPZV<1>, ZPZV<379>, ZPZV<2»; }; // NOLINT
04336
              template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387»; }; // NOLINT
template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<266>, ZPZV<2»; }; // NOLINT
04337
               template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<387»; }; // NOLINT
04339
              template<> struct ConwayPolynomial<389, 6> { using ZPZ = aerobus::zpz<389>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<339>, ZPZV<255>, ZPZV<2»; }; // NOLINT
04340
              template<> struct ConwayPolynomial<389, 7> { using ZPZ = aerobus::zpz<389>; using type =
        POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<287; // NOLINT template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::zpz<389>; using type =
04341
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<351>, ZPZV<19>, ZPZV<290>, ZPZV<22»; };
         template<> struct ConwayPolynomial<389, 9> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<258>, ZPZV<387»;</pre>
04342
         }; // NOLINT
               template<> struct ConwayPolynomial<397, 1> { using ZPZ = aerobus::zpz<397>; using type =
         POLYV<ZPZV<1>, ZPZV<392»; }; // NOLINT
              template<> struct ConwayPolynomial<397, 2> { using ZPZ = aerobus::zpz<397>; using type =
        POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 3> { using ZPZ = aerobus::zpz<397>; using type =
04345
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<392»; }; // NOLINT template<> struct ConwayPolynomial<397, 4> { using ZPZ = aerobus::zpz<397>; using type =
04346
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
04347
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392»; }; // NOLINT
              template<> struct ConwayPolynomial<397, 6> { using ZPZ = aerobus::zpz<397>; using type =
04348
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<274>, ZPZV<287>, ZPZV<287>; using type stemplate<> struct ConwayPolynomial<397, 7> { using ZPZ = aerobus::zpz<397>; using type stemplate<> struct ConwayPolynomial<397, 7> { using ZPZ = aerobus::zpz<397>; using type stemplate
04349
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<129; };
               template<> struct ConwayPolynomial<397, 8> { using ZPZ = aerobus::zpz<397>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<375>, ZPZV<255>, ZPZV<203>, ZPZV<5»; }; //
04351
              template<> struct ConwayPolynomial<397, 9> { using ZPZ = aerobus::zpz<397>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<166>, ZPZV<252>, ZPZV<392»;
         }; // NOLINT
    template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
04352
         POLYV<ZPZV<1>, ZPZV<398»; }; // NOLINT
               template<> struct ConwayPolynomial<401, 2> { using ZPZ = aerobus::zpz<401>; using type =
04353
         POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
04354
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<398»; }; // NOLINT template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<401, 5> { using ZPZ = aerobus::zpz<401>; using type =
04356
         template<> struct ConwayPolynomial<401, 6> { using ZPZ = aerobus::zpz<401>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<115>, ZPZV<81>, ZPZV<51>, ZPZV<3»; }; // NOLINT
04357
04358
               template<> struct ConwayPolynomial<401, 7> { using ZPZ = aerobus::zpz<401>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<398»; }; // NOLINT template<> struct ConwayPolynomial<401, 8> { using ZPZ = aerobus::zpz<401>; using type =
04359
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<380>, ZPZV<113>, ZPZV<164>, ZPZV<3»; }; //
         NOLINT
              template<> struct ConwayPolynomial<401, 9> { using ZPZ = aerobus::zpz<401>; using type =
04360
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<158>, ZPZV<398»;
         }; // NOLINT
04361
               template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
         POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
               template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
04362
         POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT
               template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
04363
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
04364
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<407>, ZPZV<21»; }; // NOLINT template<> struct ConwayPolynomial<409, 5> { using ZPZ = aerobus::zpz<409>; using type =
04365
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<388»; }; // NOLINT
```

```
04366
                        template<> struct ConwayPolynomial<409, 6> { using ZPZ = aerobus::zpz<409>; using type =
             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<372>, ZPZV<53>, ZPZV<364>, ZPZV<21»; }; // NOLINT template<> struct ConwayPolynomial<409, 7> { using ZPZ = aerobus::zpz<409>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<388»; }; // NOLINT template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
04368
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<256>, ZPZV<69>, ZPZV<396>, ZPZV<396, ZPZV<21»; }; //
              NOLINT
                        template<> struct ConwayPolynomial<409, 9> { using ZPZ = aerobus::zpz<409>; using type
04369
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<318>, ZPZV<211>, ZPZV<388»;
              }; // NOLINT
04370
                       template<> struct ConwayPolynomial<419, 1> { using ZPZ = aerobus::zpz<419>; using type =
             POLYV<ZPZV<1>, ZPZV<417»; }; // NOLINT
                       template<> struct ConwayPolynomial<419, 2> { using ZPZ = aerobus::zpz<419>; using type =
04371
              POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT
04372
                      template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<373>, ZPZV<2»; }; // NOLINT
04373
                       template<> struct ConwayPolynomial<419, 5> { using ZPZ = aerobus::zpz<419>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; // NOLINT
                        template<> struct ConwayPolynomial<419, 6> { using ZPZ = aerobus::zpz<419>; using type =
04375
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<411>, ZPZV<33>, ZPZV<257>, ZPZV<2»; }; // NOLINT
                      template<> struct ConwayPolynomial<419, 7> { using ZPZ = aerobus::zpz<419>; using type =
04376
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZV<0>; ZPZV<0
04377
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<234>, ZPZV<388>, ZPZV<351>, ZPZV<2»; }; //
             template<> struct ConwayPolynomial<419, 9> { using ZPZ = aerobus::zpz<419>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<386>, ZPZV<417»;</pre>
04378
              }; // NOLINT
                       template<> struct ConwayPolynomial<421, 1> { using ZPZ = aerobus::zpz<421>; using type =
04379
             POLYV<ZPZV<1>, ZPZV<419»; }; // NOLINT
                       template<> struct ConwayPolynomial<421, 2> { using ZPZ = aerobus::zpz<421>; using type =
04380
              POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT
04381
                       template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419»; }; // NOLINT
template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2»; }; // NOLINT
04382
04383
                        template<> struct ConwayPolynomial<421, 5> { using ZPZ = aerobus::zpz<421>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<419»; }; // NOLINT
04384
                      template<> struct ConwayPolynomial<421, 6> { using ZPZ = aerobus::zpz<421>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<342>, ZPZV<41>, ZPZV<2»; }; // NOLINT templated struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct conwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type struct ConwayPolynomial<421>; using type struct ConwayPolyno
04385
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<419»; };
                       template<> struct ConwayPolynomial<421, 8> { using ZPZ = aerobus::zpz<421>; using type =
              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<389>, ZPZV<32>, ZPZV<77>, ZPZV<2»; };
             template<> struct ConwayPolynomial<421, 9> { using ZPZ = aerobus::zpz<421>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<394>, ZPZV<145>, ZPZV<419»;</pre>
04387
              }; // NOLINT
04388
                        template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
              POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
04389
                       template<> struct ConwayPolynomial<431, 2> { using ZPZ = aerobus::zpz<431>; using type =
             POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<431, 3> { using ZPZ = aerobus::zpz<431>; using type =
04390
             POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<2424%; }; // NOLINT template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7»; }; // NOLINT
                       template<> struct ConwayPolynomial<431, 5> { using ZPZ = aerobus::zpz<431>; using type =
04392
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<424»; }; // NOLINT
                      template<> struct ConwayPolynomial<431, 6> { using ZPZ = aerobus::zpz<431>; using type =
04393
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<161>, ZPZV<202>, ZPZV<182>, ZPZV<182>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<431, 7> { using ZPZ = aerobus::zpz<431>; using type
04394
              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; //
04395
                       template<> struct ConwayPolynomial<431, 8> { using ZPZ = aerobus::zpz<431>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<243>, ZPZV<286>, ZPZV<115>, ZPZV<7»; }; //
              NOLINT
04396
                      template<> struct ConwayPolynomial<431, 9> { using ZPZ = aerobus::zpz<431>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<71>, ZPZV<329>, ZPZV<424*;
              }; // NOLINT
                       template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
04397
             POLYV<ZPZV<1>, ZPZV<428»; }; // NOLINT
                       template<> struct ConwayPolynomial<433, 2> { using ZPZ = aerobus::zpz<433>; using type =
04398
             POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5»; }; // NOLINT
                       template<> struct ConwayPolynomial<433, 3> { using ZPZ = aerobus::zpz<433>; using type =
04399
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428»; }; // NOLINT
                       template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
04400
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<402>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<433, 5> { using ZPZ = aerobus::zpz<433>; using type =
04401
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<428»; }; // NOLINT template<> struct ConwayPolynomial<433, 6> { using ZPZ = aerobus::zpz<433>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<244>, ZPZV<353>, ZPZV<360>, ZPZV<5»; }; // NOLINT
04402
                       template<> struct ConwayPolynomial<433, 7> { using ZPZ = aerobus::zpz<433>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<428»; };
04404
                      \texttt{template<>} \texttt{struct ConwayPolynomial<433, 8> \{ \texttt{using ZPZ = aerobus::zpz<433>; using type = aerobus::zpz<433>; using typ
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347>, ZPZV<32>, ZPZV<39>, ZPZV<5»; }; //
              NOLTNT
```

```
template<> struct ConwayPolynomial<433, 9> { using ZPZ = aerobus::zpz<433>; using type :
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<27>, ZPZV<232>, ZPZV<45>, ZPZV<428»;
       }; // NOLINT
04406
            template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
       POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
            template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
04407
       POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; }; // NOLINT
04408
            template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4243*; }; // NOLINT
template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; }; // NOLINT
template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15»; }; // NOLINT
Template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
04409
04410
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT
            template<> struct ConwayPolynomial<439, 6> { using ZPZ = aerobus::zpz<439>; using type =
04411
       POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<1>, 2PZV<324>, 2PZV<190>, 2PZV<15»; }; // NOLINT
04412
            template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 8> { using ZPZ = aerobus::zpz<439>; using type =
04413
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<359>, ZPZV<296>, ZPZV<266>, ZPZV<268>, ZPZV<15»; }; //
       template<> struct ConwayPolynomial<439, 9> { using ZPZ = aerobus::zpz<439>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<342>, ZPZV<342>, ZPZV<254>, ZPZV<424»;
       }; // NOLINT
            template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
04415
       POLYV<ZPZV<1>, ZPZV<441»; }; // NOLINT
            template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
       POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; }; // NOLINT
04417
            template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
04418
       POLYVCZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<383>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<443>; using type =
04419
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT
04420
           template<> struct ConwayPolynomial<443, 6> { using ZPZ = aerobus::zpz<443>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<298>, ZPZV<218>, ZPZV<41>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<443, 7> { using ZPZ = aerobus::zpz<443>; using type =
04421
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<41»; }; // NOLINT template<> struct ConwayPolynomial<443, 8> { using ZPZ = aerobus::zpz<443>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<217>, ZPZV<290>, ZPZV<2»; };
04423
           template<> struct ConwayPolynomial<443, 9> { using ZPZ = aerobus::zpz<443>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<125>, ZPZV<109, ZPZV<441»;
       }; // NOLINT
04424
            template<> struct ConwayPolynomial<449, 1> { using ZPZ = aerobus::zpz<449>; using type =
       POLYV<ZPZV<1>, ZPZV<446»; }; // NOLINT
04425
            template<> struct ConwayPolynomial<449, 2> { using ZPZ = aerobus::zpz<449>; using type =
       POLYV<ZPZV<1>, ZPZV<444>, ZPZV<3»; }; // NOLINT
04426
           template<> struct ConwayPolynomial<449, 3> { using ZPZ = aerobus::zpz<449>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446»; }; // NOLINT template<> struct ConwayPolynomial<449, 4> { using ZPZ = aerobus::zpz<449>; using type =
04427
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<449, 5> { using ZPZ = aerobus::zpz<449>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<446»; }; // NOLINT
04429
            template<> struct ConwayPolynomial<449, 6> { using ZPZ = aerobus::zpz<449>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<69>, ZPZV<439>, i; // NOLINT template<> struct ConwayPolynomial<449, 7> { using ZPZ = aerobus::zpz<449>; using type
04430
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<446»; }; // NOLINT
           template<> struct ConwayPolynomial<449, 8> { using ZPZ = aerobus::zpz<449>; using type =
04431
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361>, ZPZV<348>, ZPZV<124>, ZPZV<3»; }; //
       NOLINT
           template<> struct ConwayPolynomial<449, 9> { using ZPZ = aerobus::zpz<449>; using type =
04432
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<226>, ZPZV<9>, ZPZV<446»; };
       // NOLINT
            template<> struct ConwayPolynomial<457, 1> { using ZPZ = aerobus::zpz<457>; using type =
       POLYV<ZPZV<1>, ZPZV<444»; }; // NOLINT
04434
            template<> struct ConwayPolynomial<457, 2> { using ZPZ = aerobus::zpz<457>; using type =
       POLYV<ZPZV<1>, ZPZV<454>, ZPZV<453»; }; // NOLINT template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
04435
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444*; }; // NOLINT
            template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
04436
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<407>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<457, 5> { using ZPZ = aerobus::zpz<457>; using type =
04437
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44, ZPZV<444*; }; // NOLINT template<> struct ConwayPolynomial<457, 6> { using ZPZ = aerobus::zpz<457>; using type =
04438
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<205>, ZPZV<266>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<457, 7> { using ZPZ = aerobus::zpz<457>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<444»; };
04440
           template<> struct ConwayPolynomial<457, 8> { using ZPZ = aerobus::zpz<457>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<365>, ZPZV<296>, ZPZV<412>, ZPZV<13»; }; //
       NOLINT
            template<> struct ConwayPolynomial<457, 9> { using ZPZ = aerobus::zpz<457>; using type =
04441
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<35, ZPZV<354>, ZPZV<844, ZPZV<444*;
       }; // NOLINT
04442
            template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
       POLYY<ZPZV<1>, ZPZV<459»; }; // NOLINT template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
       POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<461, 3> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459»; }; // NOLINT template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
04445
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
04446
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<459»; }; // NOLINT
           template<> struct ConwayPolynomial<461, 6> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<439>, ZPZV<432>, ZPZV<329>, ZPZV<2»; }; // NOLINI
          template<> struct ConwayPolynomial<461, 7> { using ZPZ = aerobus::zpz<461>; using type =
04448
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<459»; }; // NOLINT
04449
          template<> struct ConwayPolynomial<461, 8> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<449>, ZPZV<321>, ZPZV<2»; }; //
           template<> struct ConwayPolynomial<461, 9> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<510>, ZPZV<216>, ZPZV<459»;
      }; // NOLINT
04451
           template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<460»; }; // NOLINT
           template<> struct ConwayPolynomial<463, 2> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<460»; }; // NOLINT template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
04454
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
04455
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<460»; }; // NOLINT
           template<> struct ConwayPolynomial<463, 6> { using ZPZ = aerobus::zpz<463>; using type =
04456
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<462>, ZPZV<51>, ZPZV<110>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<463, 7> { using ZPZ = aerobus::zpz<463>; using type :
04457
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<460»; }; // NOLINT
          template<> struct ConwayPolynomial<463, 8> { using ZPZ = aerobus::zpz<463>; using type =
04458
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3414>, ZPZV<396>, ZPZV<39; }; //
          template<> struct ConwayPolynomial<463, 9> { using ZPZ = aerobus::zpz<463>; using type =
04459
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<433>, ZPZV<433>, ZPZV<227>, ZPZV<460»;
      }; // NOLINT
      template<> struct ConwayPolynomial<467, 1> { using ZPZ = aerobus::zpz<467>; using type = POLYV<ZPZV<1>, ZPZV<465»; }; // NOLINT
04460
           template<> struct ConwayPolynomial<467, 2> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; }; // NOLINT
04462
           template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; }; // NOLINT
04463
           template<> struct ConwayPolynomial<467, 5> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
04465
          template<> struct ConwayPolynomial<467, 6> { using ZPZ = aerobus::zpz<467>; using type =
      template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type =
04466
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<465»; }; // NOLINT
           template<> struct ConwayPolynomial<467, 8> { using ZPZ = aerobus::zpz<467>; using type
04467
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<318>, ZPZV<413>, ZPZV<289>, ZPZV<2»; }; //
      NOLINT
      04468
      }; // NOLINT
           template<> struct ConwayPolynomial<479, 1> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<466»; }; // NOLINT
           template<> struct ConwayPolynomial<479, 2> { using ZPZ = aerobus::zpz<479>; using type =
04470
      POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; }; // NOLINT
          template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
04471
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466»; }; // NOLINT template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
04472
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13»; };
                                                                    // NOLINT
04473
          template<> struct ConwayPolynomial<479, 5> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<466»; }; // NOLINT
04474
           template<> struct ConwayPolynomial<479, 6> { using ZPZ = aerobus::zpz<479>; using type =
      POLYY<ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<243>, ZPZV<243>, ZPZV<2433, ZPZV<334>, ZPZV<13s; }; // NOLINT template<> struct ConwayPolynomial<479, 7> { using ZPZ = aerobus::zpz<479>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<46%; };
           template<> struct ConwayPolynomial<479, 8> { using ZPZ = aerobus::zpz<479>; using type
04476
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<247>, ZPZV<440>, ZPZV<17>, ZPZV<13»; }; //
      NOLINT
04477
          template<> struct ConwayPolynomial<479, 9> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<165>, ZPZV<466»; };
04478
          template<> struct ConwayPolynomial<487, 1> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<484»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 2> { using ZPZ = aerobus::zpz<487>; using type =
04479
      POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
04480
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; }; // NOLINT
      template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<483>, ZPZV<3»; }; // NOLINT
04482
          template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<484»; }; // NOLINT
04483
          template<> struct ConwayPolynomial<487, 6> { using ZPZ = aerobus::zpz<487>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<427>, ZPZV<185>, ZPZV<3»; };
            template<> struct ConwayPolynomial<487, 7> { using ZPZ = aerobus::zpz<487>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<484»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 8> { using ZPZ = aerobus::zpz<487>; using type =
04485
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<249>, ZPZV<137>, ZPZV<3»; }; //
       NOLTNT
       template<> struct ConwayPolynomial<487, 9> { using ZPZ = aerobus::zpz<487>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<271>, ZPZV<4447>, ZPZV<484»;
04486
       }; // NOLINT
04487
           template<> struct ConwayPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<489»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
04488
       POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type =
04490
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360, ZPZV<2x; }; // NOLINT template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
04491
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT
04492
           template<> struct ConwayPolynomial<491, 6> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<369>, ZPZV<402>, ZPZV<125>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<491, 7> { using ZPZ = aerobus::zpz<491>; using type =
04493
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 8> { using ZPZ = aerobus::zpz<491>; using type =
04494
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378>, ZPZV<372>, ZPZV<216>, ZPZV<2»; }; //
           template<> struct ConwayPolynomial<491, 9> { using ZPZ = aerobus::zpz<491>; using type =
04495
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<453>, ZPZV<489»;
       }; // NOLINT
04496
           template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
       POLYV<ZPZV<1>, ZPZV<492»; }; // NOLINT
04497
            template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
       POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; }; // NOLINT
04498
           template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; ); // NOLINT
template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
04499
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<492»; }; // NOLINT
           template<> struct ConwayPolynomial<499, 6> { using ZPZ = aerobus::zpz<499>; using type =
04501
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<407>, ZPZV<191>, ZPZV<78>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<499, 7> { using ZPZ = aerobus::zpz<499>; using type =
04502
       POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<492»; }; // NOLINT
           template<> struct ConwayPolynomial<499, 8> { using ZPZ = aerobus::zpz<499>; using type =
04503
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<288>, ZPZV<309>, ZPZV<200>, ZPZV<7»; }; //
       NOLINT
04504
           template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<491>, ZPZV<222>, ZPZV<492»;
       }; // NOLINT
04505
            template<> struct ConwavPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
       POLYV<ZPZV<1>, ZPZV<498»; }; // NOLINT
            template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
       POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
04507
      POLYV<ZPZV<1>, ZPZV<0, ZPZV<2>, ZPZV<498»; }; // NOLINT template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<32>, ZPZV<5»; }; // NOLINT
04508
           template<> struct ConwayPolynomial<503, 5> { using ZPZ = aerobus::zpz<503>; using type =
04509
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<498»; }; // NOLINT
04510
            template<> struct ConwayPolynomial<503, 6> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<380>, ZPZV<292>, ZPZV<255>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<503, 7> { using ZPZ = aerobus::zpz<503>; using type
04511
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<498»; }; // NOLINT
           template<> struct ConwayPolynomial<503, 8> { using ZPZ = aerobus::zpz<503>, using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<441>, ZPZV<203>, ZPZV<316>, ZPZV<5»; }; //
       NOLINT
04513
       template<> struct ConwayPolynomial<503, 9> { using ZPZ = aerobus::zpz<503>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<158>, ZPZV<337>, ZPZV<498»;</pre>
       }; // NOLINT
04514
            template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<507»; }; // NOLINT
04515
           template<> struct ConwayPolynomial<509, 2> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
04516
       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2»; }; // NOLINT
04518
            template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
04519
           template<> struct ConwayPolynomial<509, 6> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<350>, ZPZV<232>, ZPZV<41>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<509,
                                                           7> { using ZPZ = aerobus::zpz<509>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5), ZPZV<5, ZPZV<50, ZPZV<50, ZPZV<50; ; // NOLII template<> struct ConwayPolynomial<509, 8> { using ZPZ = aerobus::zpz<509>; using type =
04521
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<420>, ZPZV<473>, ZPZV<382>, ZPZV<2»; }; //
       NOLINT
04522
           template<> struct ConwayPolynomial<509, 9> { using ZPZ = aerobus::zpz<509>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3, Z
04523
                 template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
          POLYV<ZPZV<1>, ZPZV<518»; }; // NOLINT
04524
                 template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
          POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518»; }; // NOLINT
                 template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
04526
          template<> struct ConwayPolynomial<521, 5> { using ZPZ = aerobus::zpz<521>; using type =
04527
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<518»; }; // NOLINT
04528
                 template<> struct ConwayPolynomial<521, 6> { using ZPZ = aerobus::zpz<521>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<315>, ZPZV<153>, ZPZV<280>, ZPZV<3»; }; // NOLIN
04529
                 template<> struct ConwayPolynomial<521, 7> { using ZPZ = aerobus::zpz<521>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<518»; }; // NOLINT template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type =
04530
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<462>, ZPZV<407>, ZPZV<312>, ZPZV<3»; }; //
04531
                 template<> struct ConwayPolynomial<521, 9> { using ZPZ = aerobus::zpz<521>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<181>, ZPZV<483>, ZPZV<518»;
          }; // NOLINT
04532
                 template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
          POLYV<ZPZV<1>, ZPZV<521»; }; // NOLINT
04533
                  template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
          POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
04534
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
04535
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
04536
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<521»; }; // NOLINT
                  template<> struct ConwayPolynomial<523, 6> { using ZPZ = aerobus::zpz<523>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<475>, ZPZV<475>, ZPZV<371>, ZPZV<2»; }; // NOLINI
04538
                  template<> struct ConwayPolynomial<523, 7> { using ZPZ = aerobus::zpz<523>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<521»; };
                 template<> struct ConwayPolynomial<523, 8> { using ZPZ = aerobus::zpz<523>; using type =
04539
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<518>, ZPZV<184>, ZPZV<380>, ZPZV<2»; }; //
          NOLINT
                 template<> struct ConwayPolynomial<523, 9> { using ZPZ = aerobus::zpz<523>; using type =
04540
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<342>, ZPZV<345>, ZPZV<521»;
          }; // NOLINT
                  template<> struct ConwayPolynomial<541, 1> { using ZPZ = aerobus::zpz<541>; using type =
04541
          POLYV<ZPZV<1>, ZPZV<539»; }; // NOLINT
                  template<> struct ConwayPolynomial<541, 2> { using ZPZ = aerobus::zpz<541>; using type =
          POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; }; // NOLINT
04543
                 template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539»; }; // NOLINT
template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
04544
04545
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<539»; }; // NOLINT
04546
                  template<> struct ConwayPolynomial<541, 6> { using ZPZ = aerobus::zpz<541>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<239>, ZPZV<320>, ZPZV<69>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<541, 7> { using ZPZ = aerobus::zpz<541>; using type =
04547
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<539»; }; // NOLINT
                  template<> struct ConwayPolynomial<541, 8> { using ZPZ = aerobus::zpz<541>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<376>, ZPZV<108>, ZPZV<113>, ZPZV<2*; }; //
04549
                  template<> struct ConwayPolynomial<541, 9> { using ZPZ = aerobus::zpz<541>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<340>, ZPZV<340>, ZPZV<339»;
          }; // NOLINT
04550
                  template<> struct ConwayPolynomial<547, 1> { using ZPZ = aerobus::zpz<547>; using type =
          POLYV<ZPZV<1>, ZPZV<545»; }; // NOLINT
04551
                 template<> struct ConwayPolynomial<547, 2> { using ZPZ = aerobus::zpz<547>; using type =
          POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; }; // NOLINT
04552
                 template<> struct ConwayPolynomial<547, 3> { using ZPZ = aerobus::zpz<547>; using type =
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<4>, ZPZV<545*; }; // NOLINT template<> struct ConwayPolynomial<547, 4> { using ZPZ = aerobus::zpz<547>; using type =
04553
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<334>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<547, 5> { using ZPZ = aerobus::zpz<547>; using type =
04554
           \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<2>, \verb"ZPZV<545"; \verb"}; \verb"// NOLINT" | 
04555
                 template<> struct ConwayPolynomial<547, 6> { using ZPZ = aerobus::zpz<547>; using type =
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<2»; }; // NOLINT
                                                                                        7> { using ZPZ = aerobus::zpz<547>; using type
04556
                 template<> struct ConwayPolynomial<547,
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<545»; }; // NOLINT
                 template<> struct ConwayPolynomial<547, 8> { using ZPZ = aerobus::zpz<547>; using type
04557
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2»; }; //
          NOLINT
04558
          template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<263>, ZPZV<545»;</pre>
          }; // NOLINT
                 template<> struct ConwayPolynomial<557, 1> { using ZPZ = aerobus::zpz<557>; using type =
          POLYV<ZPZV<1>, ZPZV<555»; }; // NOLINT
                template<> struct ConwayPolynomial<557, 2> { using ZPZ = aerobus::zpz<557>; using type =
04560
         POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
04561
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<555»; }; // NOLINT
      template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<557, 5> { using ZPZ = aerobus::zpz<557>; using type =
04563
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<555»; }; // NOLINT
      template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<20>, ZPZV<202>, ZPZV<292>, ZPZV<253>, ZPZV<2»; }; // NOLINT
04564
            template<> struct ConwayPolynomial<557, 7> { using ZPZ = aerobus::zpz<557>; using type
04565
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<555»; }; // NOLINT template<> struct ConwayPolynomial<557, 8> { using ZPZ = aerobus::zpz<557>; using type =
04566
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<384>, ZPZV<113>, ZPZV<2»; }; //
       NOLINT
           template<> struct ConwayPolynomial<557, 9> { using ZPZ = aerobus::zpz<557>; using type =
04567
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<434>, ZPZV<434>, ZPZV<555»;
       }; // NOLINT
04568
           template<> struct ConwayPolynomial<563, 1> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<561»; }; // NOLINT
           template<> struct ConwayPolynomial<563, 2> { using ZPZ = aerobus::zpz<563>; using type =
04569
       POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; }; // NOLINT template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
04571
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<20>, ZPZV<399>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<563, 5> { using ZPZ = aerobus::zpz<563>; using type =
04572
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66, ZPZV<561»; }; // NOLINT
            template<> struct ConwayPolynomial<563, 6> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<303>, ZPZV<246>, ZPZV<2»; }; // NOLINI
04574
           template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<561»; }; // NOLINT
04575
           template<> struct ConwayPolynomial<563, 8> { using ZPZ = aerobus::zpz<563>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<176>, ZPZV<509>, ZPZV<2*; }; //
      NOLINT
           template<> struct ConwayPolynomial<563, 9> { using ZPZ = aerobus::zpz<563>; using type =
04576
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<15>, ZPZV<19>, ZPZV<561»; };
       // NOLINT
04577
           template<> struct ConwayPolynomial<569, 1> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<566»; }; // NOLINT
            template<> struct ConwayPolynomial<569, 2> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
04580
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<569, 5> { using ZPZ = aerobus::zpz<569>; using type =
04581
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT
04582
           template<> struct ConwayPolynomial<569, 6> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<50>, ZPZV<263>, ZPZV<480>, ZPZV<3»; }; // NOLINT
04583
           template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<566»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 8> { using ZPZ = aerobus::zpz<569>; using type =
04584
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<527>, ZPZV<173>, ZPZV<241>, ZPZV<241>, ZPZV<3»; }; //
04585
           template<> struct ConwayPolynomial<569, 9> { using ZPZ = aerobus::zpz<569>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<478>, ZPZV<566>, ZPZV<566»;
       }; // NOLINT
04586
            template<> struct ConwayPolynomial<571, 1> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<568»; }; // NOLINT
           template<> struct ConwayPolynomial<571, 2> { using ZPZ = aerobus::zpz<571>; using type =
04587
      POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; }; // NOLINT
04588
            template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<568»; ); // NOLINT
template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3»; }; // NOLINT
04589
            template<> struct ConwayPolynomial<571, 5> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<568»; }; // NOLINT
04591
           template<> struct ConwayPolynomial<571, 6> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<221>, ZPZV<295>, ZPZV<33>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<571, 7> { using ZPZ = aerobus::zpz<571>; using type =
04592
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<568»; }; // NOLINT
            template<> struct ConwayPolynomial<571, 8> { using ZPZ = aerobus::zpz<571>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<363>, ZPZV<119>, ZPZV<371>, ZPZV<37»; }; //
      template<> struct ConwayPolynomial<571, 9> { using ZPZ = aerobus::zpz<571>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<545>, ZPZV<5179>, ZPZV<568»;</pre>
04594
       }; // NOLINT
           template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<572»; }; // NOLINT
04596
           template<> struct ConwayPolynomial<577, 2> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
04597
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572»; }; // NOLINT
           template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
04598
      POLYV<ZPZV<1>, ZPZV<4>), ZPZV<494>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<577, 5> { using ZPZ = aerobus::zpz<577>; using type =
04599
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<572»; }; // NOLINT template<> struct ConwayPolynomial<577, 6> { using ZPZ = aerobus::zpz<577>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<450>, ZPZV<25>, ZPZV<283>, ZPZV<5»; }; // NOLINT
04600
```

```
template<> struct ConwayPolynomial<577, 7> { using ZPZ = aerobus::zpz<577>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<572»; }; // NOLINT template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type =
04602
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<3»; }; //
      NOLINT
           template<> struct ConwayPolynomial<577, 9> { using ZPZ = aerobus::zpz<577>; using type =
04603
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<576>, ZPZV<449>, ZPZV<572»;
      }; // NOLINT
          template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
04604
      POLYV<ZPZV<1>, ZPZV<585»; }; // NOLINT
           template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
04605
      POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<587, 3> { using ZPZ = aerobus::zpz<587>; using type =
04606
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585»; }; // NOLINT
04607
          template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<444>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<587, 5> { using ZPZ = aerobus::zpz<587>; using type =
04608
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<585»; }; // NOLINT
           template<> struct ConwayPolynomial<587, 6> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<204>, ZPZV<121>, ZPZV<226>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type =
04610
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<585»; }; // NOLINT
          template<> struct ConwayPolynomial<587, 8> { using ZPZ = aerobus::zpz<587>; using type =
04611
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<492>, ZPZV<44>, ZPZV<91>, ZPZV<91>, ZPZV<2»; };
      NOLINT
04612
           template<> struct ConwayPolynomial<587, 9> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<585»;
      }; // NOLINT
04613
           template<> struct ConwayPolynomial<593, 1> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<590»; }; // NOLINT
           template<> struct ConwayPolynomial<593, 2> { using ZPZ = aerobus::zpz<593>; using type =
04614
      POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<593, 3> { using ZPZ = aerobus::zpz<593>; using type =
04615
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<590»; }; // NOLINT
           template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
04616
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<593, 5> { using ZPZ = aerobus::zpz<593>; using type =
04617
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<590»; }; // NOLINT
04618
           template<> struct ConwayPolynomial<593, 6> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<345>, ZPZV<478>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<593, 7> { using ZPZ = aerobus::zpz<593>; using type =
04619
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<590»; }; // NOLINT template<> struct ConwayPolynomial<593, 8> { using ZPZ = aerobus::zpz<593>; using type =
04620
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<350>, ZPZV<291>, ZPZV<495>, ZPZV<3»; }; //
04621
           template<> struct ConwayPolynomial<593, 9> { using ZPZ = aerobus::zpz<593>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<590»;
      }; // NOLINT
04622
           template<> struct ConwayPolynomial<599, 1> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<592»; }; // NOLINT
04623
           template<> struct ConwayPolynomial<599, 2> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
04624
           template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<592»; }; // NOLINT template<> struct ConwayPolynomial<599, 4> { using ZPZ = aerobus::zpz<599>; using type =
04625
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7»; };
                                                                      // NOLINT
           template<> struct ConwayPolynomial<599, 5> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<592»; }; // NOLINT
           template<> struct ConwayPolynomial<599, 6> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7»; }; // NOLINT
04628
          template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<592»; }; // NOLINT
           template<> struct ConwayPolynomial<599, 8> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<440>, ZPZV<37>, ZPZV<124>, ZPZV<7»; };
      template<> struct ConwayPolynomial<599, 9> { using ZPZ = aerobus::zpz<599>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<314>, ZPZV<114>, ZPZV<98>, ZPZV<592»;</pre>
04630
      }; // NOLINT
           template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type =
04631
      POLYV<ZPZV<1>, ZPZV<594»; }; // NOLINT
           template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<601, 3> { using ZPZ = aerobus::zpz<601>; using type =
04633
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<594»; }; // NOLINT template<> struct ConwayPolynomial<601, 4> { using ZPZ = aerobus::zpz<601>; using type =
04634
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<601, 5> { using ZPZ = aerobus::zpz<601>; using type =
04635
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<594»; }; // NOLINT
04636
           template<> struct ConwayPolynomial<601, 6> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<440>, ZPZV<49>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<601, 7> { using ZPZ = aerobus::zpz<601>; using type =
04637
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<594»; };
           template<> struct ConwayPolynomial<601, 8> { using ZPZ = aerobus::zpz<601>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<550>, ZPZV<241>, ZPZV<490>, ZPZV<47»; }; //
      template<> struct ConwayPolynomial<601, 9> { using ZPZ = aerobus::zpz<601>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<487>, ZPZV<487>, ZPZV<590>, ZPZV<594»;</pre>
04639
```

```
}; // NOLINT
04640
               template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
         POLYY<ZPZV<1>, ZPZV<604»; }; // NOLINT template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
04641
         POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
04642
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<604»; }; // NOLINT
               template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
04643
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<607, 5> { using ZPZ = aerobus::zpz<607>; using type =
04644
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<604»; }; // NOLINT
               template<> struct ConwayPolynomial<607, 6> { using ZPZ = aerobus::zpz<607>; using type =
04645
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<45>, ZPZV<478>, ZPZV<478>; // NOLINT template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<604»; }; // NOLINT
04647
               template<> struct ConwayPolynomial<607, 8> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<468>, ZPZV<35>, ZPZV<449>, ZPZV<3»; }; //
         NOLINT
               template<> struct ConwayPolynomial<607, 9> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<444>, ZPZV<129>, ZPZV<604»;
         }; // NOLINT
04649
               template<> struct ConwayPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<611»; }; // NOLINT
               template<> struct ConwayPolynomial6613, 2> { using ZPZ = aerobus::zpz<613>; using type =
04650
         POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<611»; }; // NOLINT template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
04652
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
04653
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<611»; }; // NOLINT
04654
               template<> struct ConwayPolynomial<613, 6> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<609>, ZPZV<595>, ZPZV<601>, ZPZV<2»; }; // NOLINT
04655
              template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<611»; }; // NOLINT template<> struct ConwayPolynomial<613, 8> { using ZPZ = aerobus::zpz<613>; using type =
04656
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<489>, ZPZV<57>, ZPZV<539>, ZPZV<2»; }; //
04657
               template<> struct ConwayPolynomial<613, 9> { using ZPZ = aerobus::zpz<613>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<513>, ZPZV<536>, ZPZV<611»;
         }; // NOLINT
04658
               template<> struct ConwayPolynomial<617, 1> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<614»; }; // NOLINT
               template<> struct ConwayPolynomial<617, 2> { using ZPZ = aerobus::zpz<617>; using type =
04659
         POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; }; // NOLINT
04660
               template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<614»; }; // NOLINT
template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<617, 5> { using ZPZ = aerobus::zpz<617>; using type =
04661
04662
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<614»; }; // NOLINT
               template<> struct ConwayPolynomial<617, 6> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<318>, ZPZV<595>, ZPZV<310>, ZPZV<3»; }; // NOLINT
        template<> struct ConwayPolynomial<617, 7> { using ZPZ = aerobus::zpz<617>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<614%; }; // NOLINT
04664
               template<> struct ConwayPolynomial<617, 8> { using ZPZ = aerobus::zpz<617>; using type =
04665
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<519>, ZPZV<501>, ZPZV<155>, ZPZV<3»; }; //
         template<> struct ConwayPolynomial<617, 9> { using ZPZ = aerobus::zpz<617>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<388>, ZPZV<543>, ZPZV<614»;
         }; // NOLINT
04667
               template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<617»; }; // NOLINT
               template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; }; // NOLINT
04669
              template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617»; }; // NOLINT template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
04670
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<619, 5> { using ZPZ = aerobus::zpz<619>; using type =
04671
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<617»; }; // NOLINT
04672
              template<> struct ConwayPolynomial<619, 6> { using ZPZ = aerobus::zpz<619>; using type =
        POLYV<ZPZVv1>, ZPZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<468>, ZPZV<347>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<619, 7> { using ZPZ = aerobus::zpz<619>; using type =
04673
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<5, ZPZV<6>, 
                                                                                                                                         // NOLINT
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<416>, ZPZV<383>, ZPZV<225>, ZPZV<2»; }; //
04675
              template<> struct ConwayPolynomial<619, 9> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<310>, ZPZV<617»;
         }; // NOLINT
               template<> struct ConwayPolynomial<631, 1> { using ZPZ = aerobus::zpz<631>; using type =
        POLYV<ZPZV<1>, ZPZV<628»; }; // NOLINT
04677
              template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
        POLYV<2P2V<1>, ZPZV<629>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<631, 3> { using ZPZ = aerobus::zpz<631>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
```

```
04679
            template<> struct ConwayPolynomial<631, 4> { using ZPZ = aerobus::zpz<631>; using type =
       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<3%; }; // NOLINT template<> struct ConwayPolynomial<631, 5> { using ZPZ = aerobus::zpz<631>; using type =
04680
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
      template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<516>, ZPZV<541>, ZPZV<106>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<631, 7> { using ZPZ = aerobus::zpz<631>; using type =
04681
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<628»; };
           template<> struct ConwayPolynomial<631, 8> { using ZPZ = aerobus::zpz<631>; using type =
04683
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<3»; }; //
       NOLINT
           template<> struct ConwayPolynomial<631, 9> { using ZPZ = aerobus::zpz<631>; using type =
04684
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<296>, ZPZV<413>, ZPZV<628»;
       }; // NOLINT
04685
           template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
       POLYV<ZPZV<1>, ZPZV<638»; }; // NOLINT
           template<> struct ConwayPolynomial<641, 2> { using ZPZ = aerobus::zpz<641>; using type =
04686
       POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<638»; }; // NOLINT template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<629>, ZPZV<3»; }; // NOLINT
04689
           template<> struct ConwayPolynomial<641, 5> { using ZPZ = aerobus::zpz<641>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<638»; }; // NOLINT
            template<> struct ConwayPolynomial<641, 6> { using ZPZ = aerobus::zpz<641>; using type =
04690
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<557>, ZPZV<294>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type =
04691
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<638»; }; // NOLINT template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>; using type =
04692
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<3»; }; //
       NOLINT
04693
            template<> struct ConwayPolynomial<641, 9> { using ZPZ = aerobus::zpz<641>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<141>, ZPZV<638»;
       }; // NOLINT
04694
           template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
       POLYV<ZPZV<1>, ZPZV<632»; }; // NOLINT
           template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
04695
       POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11»; }; // NOLINT
04696
            template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<632»; }; // NOLINT template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
04697
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<600>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<643, 5> { using ZPZ = aerobus::zpz<643>; using type =
04698
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<632»; }; // NOLINT
            template<> struct ConwayPolynomial<643, 6> { using ZPZ = aerobus::zpz<643>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<345>, ZPZV<412>, ZPZV<293>, ZPZV<11»; }; // NOLINT
04700
           template<> struct ConwayPolynomial<643, 7> { using ZPZ = aerobus::zpz<643>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<632»; }; // NOLINT
           template<> struct ConwayPolynomial<643, 8> { using ZPZ = aerobus::zpz<643>; using type =
04701
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<573>, ZPZV<569>, ZPZV<11»; }; //
           template<> struct ConwayPolynomial<643, 9> { using ZPZ = aerobus::zpz<643>; using type =
04702
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<591>, ZPZV<475>, ZPZV<632»;
       }; // NOLINT
04703
           template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
       POLYV<ZPZV<1>, ZPZV<642»; }; // NOLINT
            template<> struct ConwayPolynomial<647, 2> { using ZPZ = aerobus::zpz<647>; using type =
       POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5»; }; // NOLINT
            template<> struct ConwayPolynomial<647, 3> { using ZPZ = aerobus::zpz<647>; using type =
04705
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<64>, ZPZV<642*; }; // NOLINT
template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5>; }; // NOLINT
template<> struct ConwayPolynomial<647, 5> { using ZPZ = aerobus::zpz<647>; using type =
04706
04707
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<642»; }; // NOLINT
04708
           template<> struct ConwayPolynomial<647, 6> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<308>, ZPZV<385>, ZPZV<642>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<647, 7> { using ZPZ = aerobus::zpz<647>; using type
04709
       POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<642»; }; // NOLINT
           template<> struct ConwayPolynomial<647, 8> { using ZPZ = aerobus::zpz<647>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6), ZPZV<603>, ZPZV<259>, ZPZV<271>, ZPZV<5»; };
04711
           template<> struct ConwayPolynomial<647, 9> { using ZPZ = aerobus::zpz<647>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<561>, ZPZV<123>, ZPZV<642»;
       }; // NOLINT
04712
            template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
       POLYV<ZPZV<1>, ZPZV<651»; }; // NOLINT
           template<> struct ConwayPolynomial<653, 2> { using ZPZ = aerobus::zpz<653>; using type =
04713
       POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2»; }; // NOLINT
04714
           template<> struct ConwayPolynomial<653, 3> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<651»; }; // NOLINT template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type =
04715
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2»; };
                                                                        // NOLINT
            template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651»; }; // NOLINT
04717
           template<> struct ConwayPolynomial<653, 6> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<45>, ZPZV<220>, ZPZV<242>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type =
04718
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<1>, ZPZV<651»; };
            template<> struct ConwayPolynomial<653, 8> { using ZPZ = aerobus::zpz<653>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<385>, ZPZV<18>, ZPZV<296>, ZPZV<2»; }; //
       NOLINT
04720
            template<> struct ConwayPolynomial<653, 9> { using ZPZ = aerobus::zpz<653>; using type =
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6), ZPZV<0>, ZPZV<0>, ZPZV<365>, ZPZV<665, ZPZV<651»;
       }; // NOLINT
            template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
04721
       POLYV<ZPZV<1>, ZPZV<657»; }; // NOLINT
           template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
04722
       POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
04723
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<657»; }; // NOLINT template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2»; }; // NOLINT
04725
            template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<657»; }; // NOLINT
       template<> struct ConwayPolynomial<559, 6> { using ZPZ = aerobus::zpz<559>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<371>, ZPZV<105>, ZPZV<223>, ZPZV<2»; }; // NOLINT
04726
04727
            template<> struct ConwayPolynomial<659, 7> { using ZPZ = aerobus::zpz<659>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<657»; }; //
04728
           template<> struct ConwayPolynomial<659, 8> { using ZPZ = aerobus::zpz<659>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<358>, ZPZV<246>, ZPZV<90>, ZPZV<2»; }; //
       NOLINT
04729
            template<> struct ConwayPolynomial<659, 9> { using ZPZ = aerobus::zpz<659>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592, ZPZV<592, ZPZV<465, ZPZV<657»;
       }; // NOLINT
04730
           template<> struct ConwayPolynomial<661, 1> { using ZPZ = aerobus::zpz<661>; using type =
       POLYV<ZPZV<1>, ZPZV<659»; }; // NOLINT
           template<> struct ConwayPolynomial<661, 2> { using ZPZ = aerobus::zpz<661>; using type =
04731
       POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2»; }; // NOLINT
04732
            template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<4>, ZPZV<659»; }; // NOLINT template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type =
04733
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
04734
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<659»; }; // NOLINT template<> struct ConwayPolynomial<661, 6> { using ZPZ = aerobus::zpz<661>; using type =
       POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<456>, ZPZV<382>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<661, 7> { using ZPZ = aerobus::zpz<661>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<659»; }; // NOLINT
       template<> struct ConwayPolynomial<661, 8> { using ZPZ = aerobus::zpz<661>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<612>, ZPZV<285>, ZPZV<72>, ZPZV<72>; };
04737
       template<> struct ConwayPolynomial<661, 9> { using ZPZ = aerobus::zpz<661>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<389>, ZPZV<389>, ZPZV<220>, ZPZV<659»;</pre>
       }; // NOLINT
04739
            template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<668»; }; // NOLINT
            template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
04740
       POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5»; }; // NOLINT
            template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
04741
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
04742
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5»; }; // NOLINT
            template<> struct ConwayPolynomial<673, 5> { using ZPZ = aerobus::zpz<673>; using type =
04743
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<668»; }; // NOLINT
           template<> struct ConwayPolynomial<673, 6> { using ZPZ = aerobus::zpz<673>; using type =
04744
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<524>, ZPZV<248>, ZPZV<35>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<673, 7> { using ZPZ = aerobus::zpz<673>; using type =
04745
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<60, ZPZV<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 8> { using ZPZ = aerobus::zpz<673>; using type =
04746
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<587>, ZPZV<302>, ZPZV<5»; }; //
04747
           template<> struct ConwayPolynomial<673, 9> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<347>, ZPZV<553>, ZPZV<668»;
       }; // NOLINT
04748
            template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
       POLYV<ZPZV<1>, ZPZV<675»; }; // NOLINT
04749
            template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
       POLYV<ZPZV<1>, ZPZV<672>, ZPZV<2»; }; // NOLINT
04750
           template<> struct ConwayPolynomial<677, 3> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<675»; }; // NOLINT template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
04751
       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<677, 5> { using ZPZ = aerobus::zpz<677>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<675»; }; // NOLINT
      template<> struct ConwayPolynomial<677, 6> { using ZPZ = aerobus::zpz<677>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446>, ZPZV<632>, ZPZV<50>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type =
04753
04754
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<675»; }; // NOLINT
            template<> struct ConwayPolynomial<677, 8> { using ZPZ = aerobus::zpz<677>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<363>, ZPZV<619>, ZPZV<6152>, ZPZV<2»; };
       NOLINT
04756
           template<> struct ConwayPolynomial<677, 9> { using ZPZ = aerobus::zpz<677>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<504>, ZPZV<504>, ZPZV<404, ZPZV<675»;
       }; // NOLINT
```

```
04757
            template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
       POLYV<ZPZV<1>, ZPZV<678»; }; // NOLINT
           template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
       POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; }; // NOLINT
            template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
04759
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<5>, ZPZV<588, }; // NOLINT template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5»; }; // NOLINT
04761
           template<> struct ConwayPolynomial<683, 5> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<678»; }; // NOLINT
           template<> struct ConwayPolynomial<683, 6> { using ZPZ = aerobus::zpz<683>; using type =
04762
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<644>, ZPZV<434, ZPZV<434, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<683, 7> { using ZPZ = aerobus::zpz<683>; using type
04763
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<678»; }; //
           template<> struct ConwayPolynomial<683, 8> { using ZPZ = aerobus::zpz<683>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<383>, ZPZV<184>, ZPZV<65>, ZPZV<5»; };
       NOT.TNT
       template<> struct ConwayPolynomial<683, 9> { using ZPZ = aerobus::zpz<683>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<444>, ZPZV<678»;
04765
      }; // NOLINT
   template<> struct ConwayPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
       POLYV<ZPZV<1>, ZPZV<688»; }; // NOLINT
           template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
04767
      POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; }; // NOLINT
04768
            template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<688»; }; // NOLINT
           template<> struct ConwayPolynomial<691, 4> { using ZPZ = aerobus::zpz<691>; using type =
04769
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; }; // NOLINT
04770
            template<> struct ConwayPolynomial<691, 5> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; }; // NOLINT

template<> struct ConwayPolynomial<691, 6> { using ZPZ = aerobus::zpz<691>; using type =
04771
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<408>, ZPZV<262>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; }; // NOLINT
       template<> struct ConwayPolynomial<691, 8> { using ZPZ = aerobus::zpz<691>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<356>, ZPZV<425>, ZPZV<321>, ZPZV<3»; }; //</pre>
       NOLINT
           template<> struct ConwayPolynomial<691, 9> { using ZPZ = aerobus::zpz<691>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<556>, ZPZV<443>, ZPZV<688»;
       }; // NOLINT
04775
           template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<699»; }; // NOLINT
            template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
04776
       POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; }; // NOLINT
04778
           template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
04779
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699»; }; // NOLINT
            template<> struct ConwayPolynomial<701, 6> { using ZPZ = aerobus::zpz<701>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<571>, ZPZV<327>, ZPZV<285>, ZPZV<2»; }; // NOLINT
04781
           template<> struct ConwayPolynomial<701, 7> { using ZPZ = aerobus::zpz<701>; using type =
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<699»; }; // NOLI template<> struct ConwayPolynomial<701, 8> { using ZPZ = aerobus::zpz<701>; using type =
                                                                                                             // NOLTNT
04782
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<619>, ZPZV<206>, ZPZV<593>, ZPZV<59; }; //
04783
           template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<459>, ZPZV<373>, ZPZV<699»;
       }; // NOLINT
04784
           template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
       POLYV<ZPZV<1>, ZPZV<707»; }; // NOLINT
            template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
       POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; }; // NOLINT
04786
           template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707»; }; // NOLINT
template<> struct ConwayPolynomial</ri>
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial</ri>
template</r>
04787
04788
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<707»; }; // NOLINT
            template<> struct ConwayPolynomial<709, 6> { using ZPZ = aerobus::zpz<709>; using type =
04789
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<669>, ZPZV<514>, ZPZV<295>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<709, 7> { using ZPZ = aerobus::zpz<709>; using type =
04790
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<707%; }; // NOLINT template<> struct ConwayPolynomial<709, 8> { using ZPZ = aerobus::zpz<709>; using type =
04791
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<689>, ZPZV<233>, ZPZV<79>, ZPZV<29; };
       NOLINT
04792
           template<> struct ConwayPolynomial<709, 9> { using ZPZ = aerobus::zpz<709>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<257>, ZPZV<2717>, ZPZV<707»;
       }; // NOLINT
04793
            template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<708»; }; // NOLINT
            template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type =
04795
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<602>, ZPZV<11»; };
               template<> struct ConwayPolynomial<719, 5> { using ZPZ = aerobus::zpz<719>; using type =
04797
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
04798
               template<> struct ConwayPolynomial<719, 6> { using ZPZ = aerobus::zpz<719>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<533>, ZPZV<591>, ZPZV<182>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<719, 7> { using ZPZ = aerobus::zpz<719>; using type
04799
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708%; }; // NOLINT
04800
               template<> struct ConwayPolynomial<719, 8> { using ZPZ = aerobus::zpz<719>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<714>, ZPZV<362>, ZPZV<244>, ZPZV<11»; }; //
              template<> struct ConwayPolynomial<719, 9> { using ZPZ = aerobus::zpz<719>; using type =
04801
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<38>, ZPZV<288>, ZPZV<560>, ZPZV<708»;
         }; // NOLINT
   template<>> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
04802
         POLYV<ZPZV<1>, ZPZV<722»; }; // NOLINT
04803
              template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
         POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
04804
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<722»; }; // NOLINT
               template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
04805
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<727, 5> { using ZPZ = aerobus::zpz<727>; using type =
04806
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<722»; }; // NOLINT

template<> struct ConwayPolynomial<727, 6> { using ZPZ = aerobus::zpz<727>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<86>, ZPZV<397>, ZPZV<672>, ZPZV<5»; }; // NOLINT
04807
               template<> struct ConwayPolynomial<727, 7> { using ZPZ = aerobus::zpz<727>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12»; };
04809
              template<> struct ConwayPolynomial<727, 8> { using ZPZ = aerobus::zpz<727>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<639>, ZPZV<671>, ZPZV<368>, ZPZV<5»; }; //
         NOLINT
              template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type =
04810
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<573>, ZPZV<502>, ZPZV<722»;
         }; // NOLINT
              template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
04811
         POLYV<ZPZV<1>, ZPZV<727»; }; // NOLINT
               template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
04812
         POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; }; // NOLINT
               template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727»; }; // NOLINT template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<733, 5> { using ZPZ = aerobus::zpz<733>; using type =
04815
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<727»; }; // NOLINT
04816
               template<> struct ConwayPolynomial<733, 6> { using ZPZ = aerobus::zpz<733>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<549>, ZPZV<151>, ZPZV<6»; }; // NOLINT
04817
               template<> struct ConwayPolynomial<733, 7> { using ZPZ = aerobus::zpz<733>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<727»; }; // NOLINT template<> struct ConwayPolynomial<733, 8> { using ZPZ = aerobus::zpz<733>; using type =
04818
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<532>, ZPZV<610>, ZPZV<142>, ZPZV<6%; }; //
         NOLINT
04819
               template<> struct ConwayPolynomial<733, 9> { using ZPZ = aerobus::zpz<733>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<337>, ZPZV<6>, ZPZV<727»; };
         // NOLINT
         template<> struct ConwayPolynomial<739, 1> { using ZPZ = aerobus::zpz<739>; using type =
POLYV<ZPZV<1>, ZPZV<736»; }; // NOLINT</pre>
04820
               template<> struct ConwayPolynomial<739, 2> { using ZPZ = aerobus::zpz<739>; using type =
04821
         POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
04822
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; }; // NOLINT template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
04823
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
04824
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<736»; }; // NOLINT
               template<> struct ConwayPolynomial<739, 6> { using ZPZ = aerobus::zpz<739>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<447>, ZPZV<625>, ZPZV<3»; }; // NOLINT
04826
              template<> struct ConwayPolynomial<739, 7> { using ZPZ = aerobus::zpz<739>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<44>, ZPZV<736»; }; // NOLINT template<> struct ConwayPolynomial<739, 8> { using ZPZ = aerobus::zpz<739>; using type =
04827
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<401>, ZPZV<169>, ZPZV<25>, ZPZV<3»; };
         NOLINT
               template<> struct ConwayPolynomial<739, 9> { using ZPZ = aerobus::zpz<739>; using type =
04828
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<616>, ZPZV<81>, ZPZV<736»;
         }; // NOLINT
04829
               template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
         POLYV<ZPZV<1>, ZPZV<738»; }; // NOLINT
               template<> struct ConwayPolynomial<743, 2> { using ZPZ = aerobus::zpz<743>; using type =
         POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; }; // NOLINT
04831
               template<> struct ConwayPolynomial<743, 3> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<738»; }; // NOLINT template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
04832
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<425>, ZPZV<425>, ZPZV<5>, 
04833
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT
04834
              template<> struct ConwayPolynomial<743, 6> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<236>, ZPZV<471>, ZPZV<88>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<743, 7> { using ZPZ = aerobus::zpz<743>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<738»; }; // NOLINT
04835
```

```
template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<279>, ZPZV<588>, ZPZV<58; }; //
      NOLINT
      template<> struct ConwayPolynomial<743, 9> { using ZPZ = aerobus::zpz<743>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<327>, ZPZV<676>, ZPZV<738»;</pre>
04837
      }; // NOLINT
04838
           template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
04839
          template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
04840
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; }; // NOLINT template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
04841
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<525>, ZPZV<3»; }; // NOLINT
04842
           template<> struct ConwayPolynomial<751, 5> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT template<> struct ConwayPolynomial<751, 6> { using ZPZ = aerobus::zpz<751>; using type =
04843
      POLYY<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<298>, ZPZV<539>, ZPZV<539>, ZPZV<53); / NoLINT template<> struct ConwayPolynomial<751, 7> { using ZPZ = aerobus::zpz<751>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<748»; };
           template<> struct ConwayPolynomial<751, 8> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<741>, ZPZV<243>, ZPZV<672>, ZPZV<672»; }; //
      NOLINT
      template<> struct ConwayPolynomial<751, 9> { using ZPZ = aerobus::zpz<751>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<70>, ZPZV<70>, ZPZV<70>, ZPZV<489>, ZPZV<70>
04846
      }; // NOLINT
04847
          template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
04848
           template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
04849
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT
           template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
04850
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; }; // NOLINT
04851
           template<> struct ConwayPolynomial<757, 5> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<755»; }; // NOLINT
      template<> struct ConwayPolynomial<757, 6> { using ZPZ = aerobus::zpz<757>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<753>, ZPZV<745>, ZPZV<2»; }; // NOLINT
04852
04853
           template<> struct ConwayPolynomial<757,
                                                        7> { using ZPZ = aerobus::zpz<75</pre>
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<755»; };
04854
          template<> struct ConwayPolynomial<757, 8> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<110>, ZPZV<509>, ZPZV<2»; }; //
      NOLINT
04855
          template<> struct ConwayPolynomial<757, 9> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<88>, ZPZV<688>, ZPZV<702>, ZPZV<755»;
      }; // NOLINT
04856
          template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
           template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
04857
      POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; }; // NOLINT
04858
           template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755»; }; // NOLINT
04859
           template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<761, 5> { using ZPZ = aerobus::zpz<761>; using type =
04860
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<755»; }; // NOLINT
           template<> struct ConwayPolynomial<761, 6> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<2PZV<1>, 2PZV<0>, ZPZV<2>, ZPZV<634>, ZPZV<597>, ZPZV<155>, ZPZV<6»; }; // NOLINT
           template<> struct ConwayPolynomial<761, 7> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<755»; }; // NOLIN template<> struct ConwayPolynomial<761, 8> { using ZPZ = aerobus::zpz<761>; using type =
04863
      POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<603>, ZPZV<144>, ZPZV<540>, ZPZV<6%; }; //
      NOLINT
           template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<317>, ZPZV<571>, ZPZV<755»;
      }; // NOLINT
04865
           template<> struct ConwayPolynomial<769, 1> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<758»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 2> { using ZPZ = aerobus::zpz<769>; using type =
04866
      POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; };
                                                   // NOLINT
           template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
04867
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 4> { using ZPZ = aerobus::zpz<769>; using type =
04868
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<769, 5> { using ZPZ = aerobus::zpz<769>; using type =
04869
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<758»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 6> { using ZPZ = aerobus::zpz<769>; using type =
04870
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<326>, ZPZV<650>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 7> { using ZPZ = aerobus::zpz<769>; using type =
04871
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<758»; }; // NOLINT
          template<> struct ConwayPolynomial<769, 8> { using ZPZ = aerobus::zpz<769>; using type
04872
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<560>, ZPZV<574>, ZPZV<632>, ZPZV<11»; }; //
04873
          template<> struct ConwayPolynomial<769, 9> { using ZPZ = aerobus::zpz<769>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<623>, ZPZV<751>, ZPZV<758»;
      }; // NOLINT
04874
          template<> struct ConwayPolynomial<773, 1> { using ZPZ = aerobus::zpz<773>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<771»; };
                      template<> struct ConwayPolynomial<773, 2> { using ZPZ = aerobus::zpz<773>; using type =
             POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
04876
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<771»; }; // NOLINT template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
04877
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2»; }; // NOLINT
04878
                      template<> struct ConwayPolynomial<773, 5> { using ZPZ = aerobus::zpz<773>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<771»; }; // NOLINT
            template<> struct ConwayPolynomial<773, 6> { using ZPZ = aerobus::zpz<773>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<91>, ZPZV<3>, ZPZV<581>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<773, 7> { using ZPZ = aerobus::zpz<773>; using type =
04879
04880
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<771»; }; // NOLII template<> struct ConwayPolynomial<773, 8> { using ZPZ = aerobus::zpz<773>; using type = 
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<484>, ZPZV<94>, ZPZV<693>, ZPZV<2»; };
                     template<> struct ConwayPolynomial<773, 9> { using ZPZ = aerobus::zpz<773>; using type =
04882
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<216>, ZPZV<574>, ZPZV<771»;
             }; // NOLINT
                      template<> struct ConwayPolynomial<787, 1> { using ZPZ = aerobus::zpz<787>; using type =
             POLYV<ZPZV<1>, ZPZV<785»; }; // NOLINT
                      template<> struct ConwayPolynomial<787, 2> { using ZPZ = aerobus::zpz<787>; using type =
04884
             POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
                      template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
04885
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785»; }; // NOLINT
                      template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2»; }; // NOLINT
04887
                      template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<785»; }; // NOLINT
04888
                     template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
            POLYVZPZVX1>, ZPZV<0>, ZPZV<0>, ZPZV<98>, ZPZV<512>, ZPZV<606>, ZPZV<2»; ); // NOLINT template<> struct ConwayPolynomial<787, 7> { using ZPZ = aerobus::zpz<787>; using type
04889
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZV<3
04890
                    template<> struct ConwayPolynomial<787, 8> { using ZPZ = aerobus::zpz<787>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<612>, ZPZV<26>, ZPZV<715>, ZPZV<2»; };
             NOLINT
             template<> struct ConwayPolynomial<787, 9> { using ZPZ = aerobus::zpz<787>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<573>, ZPZV<785»;
04891
             }; // NOLINT
                      template<> struct ConwayPolynomial<797, 1> { using ZPZ = aerobus::zpz<797>; using type =
04892
             POLYV<ZPZV<1>, ZPZV<795»; }; // NOLINT
                      template<> struct ConwayPolynomial<797, 2> { using ZPZ = aerobus::zpz<797>; using type =
04893
             POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»: }: // NOLINT
                     template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
04894
            POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type =
04895
            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<717>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<797, 5> { using ZPZ = aerobus::zpz<797>; using type =
04896
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<795»; }; // NOLINT
                      template<> struct ConwayPolynomial<797, 6> { using ZPZ = aerobus::zpz<797>; using type =
04897
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<657>, ZPZV<396>, ZPZV<71>, ZPZV<2»; }; // NOLINT
                      template<> struct ConwayPolynomial<797, 7> { using ZPZ = aerobus::zpz<797>, using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<795»; };
04899
                     template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type =
             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2»; }; //
             NOLINT
04900
                      template<> struct ConwayPolynomial<797, 9> { using ZPZ = aerobus::zpz<797>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<240>, ZPZV<599>, ZPZV<795»;
             }; // NOLINT
04901
                      template<> struct ConwayPolynomial<809, 1> { using ZPZ = aerobus::zpz<809>; using type =
             POLYV<ZPZV<1>, ZPZV<806»; }; // NOLINT
                     template<> struct ConwayPolynomial<809, 2> { using ZPZ = aerobus::zpz<809>; using type =
04902
             POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
                      template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<806»; }; // NOLINT template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
04904
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<809, 5> { using ZPZ = aerobus::zpz<809>; using type =
04905
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; }; // NOLINT
04906
                      template<> struct ConwayPolynomial<809, 6> { using ZPZ = aerobus::zpz<809>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<562>, ZPZV<75>, ZPZV<43>, ZPZV<3»; }; // NOLINT
04907
                    template<> struct ConwayPolynomial<809, 7> { using ZPZ = aerobus::zpz<809>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<806»; }; // NOLINT template<> struct ConwayPolynomial<809, 8> { using ZPZ = aerobus::zpz<809>; using type =
04908
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<593>, ZPZV<745>, ZPZV<673>, ZPZV
                     template<> struct ConwayPolynomial<809, 9> { using ZPZ = aerobus::zpz<809>; using type
04909
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<727>, ZPZV<806»;
             }; // NOLINT
04910
                      template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
             POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
                      template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
             POLYV<ZPZV<1>, ZPZV<806>, ZPZV<3»; }; // NOLINT
04912
                     template<> struct ConwayPolynomial<811, 3> { using ZPZ = aerobus::zpz<811>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<808»; }; // NOLINT template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3»; }; // NOLINT
04913
```

9.3 aerobus.h 169

```
04914
               template<> struct ConwayPolynomial<811, 5> { using ZPZ = aerobus::zpz<811>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<808»; }; // NOLINT
              template<> struct ConwayPolynomial<811, 6> { using ZPZ = aerobus::zpz<811>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<780>, ZPZV<755>, ZPZV<307>, ZPZV<3x; }; // NOLINT template<> struct ConwayPolynomial<811, 7> { using ZPZ = aerobus::zpz<811>; using type
04916
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3, ZPZV<31>; // NoLII template<> struct ConwayPolynomial<811, 8> { using ZPZ = aerobus::zpz<811>; using type =
                                                                                                                                       // NOLINT
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<3»; };
        template<> struct ConwayPolynomial<811, 9> { using ZPZ = aerobus::zpz<811>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<382>, ZPZV<200>, ZPZV<808»;
04918
         }; // NOLINT
04919
               template<> struct ConwayPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
         POLYV<ZPZV<1>, ZPZV<819»; }; // NOLINT
04920
               template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<821, 3> { using ZPZ = aerobus::zpz<821>; using type =
04921
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<819»; };
                                                                                 // NOLINT
               template<> struct ConwayPolynomial<821, 4> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<662>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
04923
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<819»; }; // NOLINT
              template<> struct ConwayPolynomial<821, 6> { using ZPZ = aerobus::zpz<821>; using type =
04924
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<160>, ZPZV<130>, ZPZV<803>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type
04925
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<819»; }; // NOLINT
04926
              template<> struct ConwayPolynomial<821, 8> { using ZPZ = aerobus::zpz<821>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<55>, ZPZV<556>, ZPZV<589>, ZPZV<58), //
         NOLINT
04927
              template<> struct ConwayPolynomial<821, 9> { using ZPZ = aerobus::zpz<821>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<650>, ZPZV<557>, ZPZV<819»;
         }; // NOLINT
04928
              template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
04929
               template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
04930
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
04931
               template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
04932
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<820»; }; // NOLINT template<> struct ConwayPolynomial<823, 6> { using ZPZ = aerobus::zpz<823>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<822>, ZPZV<616>, ZPZV<744>, ZPZV<3»; }; // NOLINT
04933
               template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<820»; };
04935
              template<> struct ConwayPolynomial<823, 8> { using ZPZ = aerobus::zpz<823>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<451>, ZPZV<437>, ZPZV<31>, ZPZV<3»; }; //
         NOLINT
               template<> struct ConwayPolynomial<823, 9> { using ZPZ = aerobus::zpz<823>; using type =
04936
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<60, ZPZV<609>, ZPZV<820»;
         }; // NOLINT
04937
               template<> struct ConwayPolynomial<827, 1> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<825»; }; // NOLINT
              template<> struct ConwayPolynomial<827, 2> { using ZPZ = aerobus::zpz<827>; using type =
04938
        POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825»; }; // NOLINT
               template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type =
04940
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<827, 5> { using ZPZ = aerobus::zpz<827>; using type =
04941
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<825»; }; // NOLINT
04942
               template<> struct ConwayPolynomial<827, 6> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<685>, ZPZV<601>, ZPZV<691>, ZPZV<2»; }; // NOLINT
04943
              template<> struct ConwayPolynomial<827, 7> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<825»; }; // NOLINT template<> struct ConwayPolynomial<827, 8> { using ZPZ = aerobus::zpz<827>; using type =
04944
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<812>, ZPZV<79>, ZPZV<32>, ZPZV<2»; };
         NOLINT
04945
               template<> struct ConwayPolynomial<827, 9> { using ZPZ = aerobus::zpz<827>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<372>, ZPZV<825»;
         }; // NOLINT
04946
               template<> struct ConwayPolynomial<829, 1> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<827»; }; // NOLINT
               template<> struct ConwayPolynomial<829, 2> { using ZPZ = aerobus::zpz<829>; using type =
04947
        POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
04948
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<827»; }; // NOLINT template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
04949
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<604>, ZPZV<604>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<829, 5> { using ZPZ = aerobus::zpz<829>; using type =
04950
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<827»; }; // NOLINT
               template<> struct ConwayPolynomial<829, 6> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<476>, ZPZV<817>, ZPZV<2»; }; // NOLINT
04952
              template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04953
```

170 File Documentation

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<468>, ZPZV<241>, ZPZV<138>, ZPZV<2*; }; //
04954
             template<> struct ConwayPolynomial<829, 9> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<621>, ZPZV<552>, ZPZV<827»;
        }; // NOLINT
04955
              template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<828»; }; // NOLINT
              template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
04956
        POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; }; // NOLINT
04957
              template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; }; // NOLINT
              template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
04958
        POLYV<ZPZV<1>, ZPZV<2>, ZPZV<609>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<828»; }; // NOLINT
04960
              template<> struct ConwayPolynomial<839, 6> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<370>, ZPZV<537>, ZPZV<23>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type
04961
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<828»; }; // NOLINT
              template<> struct ConwayPolynomial<839, 8> { using ZPZ = aerobus::zpz<839>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<553>, ZPZV<779>, ZPZV<329>, ZPZV<11»; }; //
        NOLINT
04963
              template<> struct ConwayPolynomial<839, 9> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<349>, ZPZV<349>, ZPZV<206>, ZPZV<828»;
        }; // NOLINT
04964
               template<> struct ConwayPolynomial<853, 1> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<ZPZV<1>, ZPZV<851»; }; // NOLINT
04965
              template<> struct ConwayPolynomial<853, 2> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<ZPZV<1>, ZPZV<852>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<853, 3> { using ZPZ = aerobus::zpz<853>; using type =
04966
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
04967
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<2»; }; // NOLINT
04968
             template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
        04969
              template<> struct ConwayPolynomial<853, 6> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<276>, ZPZV<276>, ZPZV<512>, ZPZV<23>; // NOLINT template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<851»; };
              template<> struct ConwayPolynomial<853, 8> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<544>, ZPZV<846>, ZPZV<118>, ZPZV<2»; }; //
        NOLINT
              template<> struct ConwayPolynomial<853, 9> { using ZPZ = aerobus::zpz<853>; using type =
04972
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<677>, ZPZV<821>, ZPZV<851»;
        }; // NOLINT
04973
              template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
        POLYV<ZPZV<1>, ZPZV<854»; }; // NOLINT
             template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
04974
        POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
04975
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; }; // NOLINT template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
        04977
              template<> struct ConwayPolynomial<857, 5> { using ZPZ = aerobus::zpz<857>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<854»; }; // NOLINT
              template<> struct ConwayPolynomial<857, 6> { using ZPZ = aerobus::zpz<857>; using type =
04978
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<824>, ZPZV<65>, ZPZV<3»; }; // NOLINT
             template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::zpz<857>; using type =
04979
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<854»; }; // NOLINT template<> struct ConwayPolynomial<857, 8> { using ZPZ = aerobus::zpz<857>; using type =
04980
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<611>, ZPZV<552>, ZPZV<494>, ZPZV<3»; }; //
        NOLINT
04981
              template<> struct ConwayPolynomial<857, 9> { using ZPZ = aerobus::zpz<857>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<719>, ZPZV<854»;
        }; // NOLINT
04982
              template<> struct ConwayPolynomial<859, 1> { using ZPZ = aerobus::zpz<859>; using type =
        POLYV<ZPZV<1>, ZPZV<857»; }; // NOLINT
              template<> struct ConwayPolynomial<859, 2> { using ZPZ = aerobus::zpz<859>; using type =
04983
        POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
04984
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type =
04985
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<859, 5> { using ZPZ = aerobus::zpz<859>; using type =
04986
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<857»; }; // NOLINT
              template<> struct ConwayPolynomial<859, 6> { using ZPZ = aerobus::zpz<859>; using type =
        POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<419>, ZPZV<646>, ZPZV<566>, ZPZV<2»; }; // NOLINT
04988
             template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpz<859>; using type =
        Template<> struct ConwayFolynomial<859, /> (using ZPZ - aerobus.:2pZ<859, using type - POLYV<ZPZV<1>, ZPZV<0>, 
04989
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<52>, ZPZV<446>, ZPZV<67>>, ZPZV<2»; }; //
        template<> struct ConwayPolynomial<859, 9> { using ZPZ = aerobus::zpz<859>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<648>, ZPZV<845>, ZPZV<857»;
        }; // NOLINT
              template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
04991
        POLYV<ZPZV<1>, ZPZV<858»; }; // NOLINT
```

9.3 aerobus.h 171

```
04992
                  template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
          POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; }; // NOLINT
04993
                 template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<858»; }; // NOLINT
template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5»; }; // NOLINT
04994
                 template<> struct ConwayPolynomial<863, 5> { using ZPZ = aerobus::zpz<863>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<858»; }; // NOLINT
                 template<> struct ConwayPolynomial<863, 6> { using ZPZ = aerobus::zpz<863>; using type =
04996
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<330>, ZPZV<62>, ZPZV<300>, ZPZV<5»; }; // NOLINT
04997
                 template<> struct ConwayPolynomial<863, 7> { using ZPZ = aerobus::zpz<863>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<858»; }; // NOLINT
                 template<> struct ConwayPolynomial<863, 8> { using ZPZ = aerobus::zpz<863>; using type =
04998
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<576>, ZPZV<576>, ZPZV<849>, ZPZV<59»; }; //
          NOLINT
04999
                 template<> struct ConwayPolynomial<863, 9> { using ZPZ = aerobus::zpz<863>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<381>, ZPZV<1>, ZPZV<858»; };
           // NOLINT
05000
                 template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
          POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
                  template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
          POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
05002
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875»; }; // NOLINT template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type =
05003
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<877, 5> { using ZPZ = aerobus::zpz<877>; using type =
05004
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<875»; }; // NOLINT template<> struct ConwayPolynomial<877, 6> { using ZPZ = aerobus::zpz<877>; using type =
05005
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<62>, ZPZV<629>, ZPZV<400>, ZPZV<855>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<877, 7> { using ZPZ = aerobus::zpz<877>; using type
05006
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<875»; };
                 template<> struct ConwayPolynomial<877, 8> { using ZPZ = aerobus::zpz<877>; using type
05007
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<767>, ZPZV<319>, ZPZV<347>, ZPZV<2»; }; //
          NOLINT
                 template<> struct ConwayPolynomial<877, 9> { using ZPZ = aerobus::zpz<877>; using type =
05008
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<770>, ZPZV<778>, ZPZV<875»;
05009
                  template<> struct ConwayPolynomial<881, 1> { using ZPZ = aerobus::zpz<881>; using type =
          POLYV<ZPZV<1>, ZPZV<878»; }; // NOLINT
05010
                 template<> struct ConwayPolynomial<881, 2> { using ZPZ = aerobus::zpz<881>; using type =
          POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
0.5011
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<878»; }; // NOLINT
                 template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<447>, ZPZV<3»; }; // NOLINT
05013
                 template<> struct ConwayPolynomial<881, 5> { using ZPZ = aerobus::zpz<881>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<878»; }; // NOLINT
05014
                 template<> struct ConwayPolynomial<881, 6> { using ZPZ = aerobus::zpz<881>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<419>, ZPZV<231>, ZPZV<3»; }; // NOLINT
05015
                 template<> struct ConwayPolynomial<881,
                                                                                      7> { using ZPZ = aerobus::zpz<881>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 ) , ZPZV | ZPZ
05016
                 template<> struct ConwayPolynomial<881, 8> { using ZPZ = aerobus::zpz<881>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<635>, ZPZV<490>, ZPZV<561>, ZPZV<561>, ZPZV<3»; }; //
          NOLINT
05017
                 template<> struct ConwayPolynomial<881, 9> { using ZPZ = aerobus::zpz<881>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<587>, ZPZV<510>, ZPZV<878»;
          }; // NOLINT
                 template<> struct ConwayPolynomial<883, 1> { using ZPZ = aerobus::zpz<883>; using type =
05018
          POLYV<ZPZV<1>, ZPZV<881»; }; // NOLINT
                 template<> struct ConwayPolynomial<883, 2> { using ZPZ = aerobus::zpz<883>; using type =
05019
          POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; }; // NOLINT
05020
                 template<> struct ConwayPolynomial<883, 3> { using ZPZ = aerobus::zpz<883>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<881»; }; // NOLINT template<> struct ConwayPolynomial<883, 4> { using ZPZ = aerobus::zpz<883>; using type =
05021
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<883, 5> { using ZPZ = aerobus::zpz<883>; using type =
05022
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<881»; }; // NOLINT
                 template<> struct ConwayPolynomial<883, 6> { using ZPZ = aerobus::zpz<883>; using type =
05023
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<879>, ZPZV<865>, ZPZV<871>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type =
05024
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
05025
                 template<> struct ConwayPolynomial<883, 8> { using ZPZ = aerobus::zpz<883>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<740>, ZPZV<762>, ZPZV<768>, ZPZV<768>, ZPZV<2»; }; //
          NOLINT
                 template<> struct ConwayPolynomial<883, 9> { using ZPZ = aerobus::zpz<883>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<50, ZPZV<557>, ZPZV<881»;
          }; // NOLINT
05027
                 template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
          POLYV<ZPZV<1>, ZPZV<882»; }; // NOLINT
                 template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
05028
          POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; }; // NOLINT
                 template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<882»; };
                                                                                             // NOLINT
                template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
05030
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<887, 5> { using ZPZ = aerobus::zpz<887>; using type =
05031
```

172 File Documentation

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<882»; };
         template<> struct ConwayPolynomial<887, 6> { using ZPZ = aerobus::zpz<887>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<775>, ZPZV<341>, ZPZV<28>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<887, 7> { using ZPZ = aerobus::zpz<887>; using type =
05033
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<882»; }; // NOLINT template<> struct ConwayPolynomial<887, 8> { using ZPZ = aerobus::zpz<887>; using type =
05034
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<781>, ZPZV<381>, ZPZV<706>, ZPZV<5»; }; //
05035
              template<> struct ConwayPolynomial<887, 9> { using ZPZ = aerobus::zpz<887>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<727>, ZPZV<345>, ZPZV<882»;
         }; // NOLINT
               template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
05036
         POLYV<ZPZV<1>, ZPZV<905»; }; // NOLINT
                template<> struct ConwayPolynomial<907, 2> { using ZPZ = aerobus::zpz<907>; using type =
         POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2»; }; // NOLINT
05038
               template<> struct ConwayPolynomial<907, 3> { using ZPZ = aerobus::zpz<907>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<905»; }; // NOLINT template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
05039
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; }; // NOLINT
05041
               template<> struct ConwayPolynomial<907, 6> { using ZPZ = aerobus::zpz<907>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<626>, ZPZV<752>, ZPZV<266>, ZPZV<265, ZPZV<29; }; // NOLINT template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type
05042
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<905»; }; // NOLINT
               template<> struct ConwayPolynomial<907, 8> { using ZPZ = aerobus::zpz<907>, using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<584>, ZPZV<518>, ZPZV<811>, ZPZV<81; }; //
         NOLINT
         template<> struct ConwayPolynomial<907, 9> { using ZPZ = aerobus::zpz<907>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<783>, ZPZV<783>, ZPZV<57>, ZPZV<905»;</pre>
05044
         }; // NOLINT
05045
                template<> struct ConwayPolynomial<911, 1> { using ZPZ = aerobus::zpz<911>; using type =
         POLYV<ZPZV<1>, ZPZV<894»; }; // NOLINT
              template<> struct ConwayPolynomial<911, 2> { using ZPZ = aerobus::zpz<911>; using type =
05046
         POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
05047
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; }; // NOLINT
               template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17»; }; // NOLINT
              template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
05049
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<894»; }; // NOLINT
05050
               template<> struct ConwayPolynomial<911, 6> { using ZPZ = aerobus::zpz<911>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<172>, ZPZV<683>, ZPZV<19>, ZPZV<17*; }; // NOLINT template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type
05051
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<894»; }; // NOLINT
05052
               template<> struct ConwayPolynomial<911, 8> { using ZPZ = aerobus::zpz<911>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<590>, ZPZV<168>, ZPZV<17»; }; //
         NOLINT
              template<> struct ConwayPolynomial<911, 9> { using ZPZ = aerobus::zpz<911>; using type =
05053
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<679>, ZPZV<616>, ZPZV<894»;
         }; // NOLINT
    template<> struct ConwayPolynomial<919, 1> { using ZPZ = aerobus::zpz<919>; using type =
05054
         POLYV<ZPZV<1>, ZPZV<912»; }; // NOLINT template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
05055
         POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; }; // NOLINT
               template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
05056
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912»; }; // NOLINT template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
05057
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<602>, ZPZY<*/>; // NOLINT
template<> struct ConwayPolynomial<919, 5> { using ZPZ = aerobus::zpz<919>; using type =
05058
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0, ZPZY<1>, ZPZY<1>, ZPZY<912»; }; // NOLINT template<> struct ConwayPolynomial<919, 6> { using ZPZ = aerobus::zpz<919>; using type =
05059
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<312>, ZPZV<817>, ZPZV<113>, ZPZV<7»; }; // NOLINT
               template<> struct ConwayPolynomial<919, 7> { using ZPZ = aerobus::zpz<919>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<912»; };
05061
              template<> struct ConwayPolynomial<919, 8> { using ZPZ = aerobus::zpz<919>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<202>, ZPZV<504>, ZPZV<7%; }; //
         NOLINT
              template<> struct ConwayPolynomial<919, 9> { using ZPZ = aerobus::zpz<919>; using type =
05062
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<410>, ZPZV<623>, ZPZV<912»;
         }; // NOLINT
05063
              template<> struct ConwayPolynomial<929, 1> { using ZPZ = aerobus::zpz<929>; using type =
         POLYV<ZPZV<1>, ZPZV<926»; }; // NOLINT
               template<> struct ConwayPolynomial<929, 2> { using ZPZ = aerobus::zpz<929>; using type =
05064
         POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<926»; }; // NOLINT template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type =
05066
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
05067
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<926»; }; // NOLINT
05068
               template<> struct ConwayPolynomial<929, 6> { using ZPZ = aerobus::zpz<929>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<805>, ZPZV<805>, ZPZV<86>, ZPZV<805, Z
05069
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<926»; }; // NOLINT template<> struct ConwayPolynomial<929, 8> { using ZPZ = aerobus::zpz<929>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699>, ZPZV<292>, ZPZV<586>, ZPZV<3»; }; //
05070
```

9.3 aerobus.h 173

```
NOLINT
           template<> struct ConwayPolynomial<929, 9> { using ZPZ = aerobus::zpz<929>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<481>, ZPZV<199>, ZPZV<926»;
           }; // NOLINT
05072
                   template<> struct ConwayPolynomial<937, 1> { using ZPZ = aerobus::zpz<937>; using type =
           POLYV<ZPZV<1>, ZPZV<932»; }; // NOLINT
                  template<> struct ConwayPolynomial<937, 2> { using ZPZ = aerobus::zpz<937>; using type =
           POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5»; }; // NOLINT
05074
                  template<> struct ConwayPolynomial<937, 3> { using ZPZ = aerobus::zpz<937>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<932»; }; // NOLINT template<> struct ConwayPolynomial<937, 4> { using ZPZ = aerobus::zpz<937>; using type =
05075
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<23>, ZPZV<585>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
05076
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<932»; }; // NOLINT
05077
                  template<> struct ConwayPolynomial<937, 6> { using ZPZ = aerobus::zpz<937>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<794>, ZPZV<727>, ZPZV<934>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<937, 7> { using ZPZ = aerobus::zpz<937>; using type =
05078
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<658>, ZPZV<26>, ZPZV<53>, ZPZV<5»: };
05080
                  template<> struct ConwayPolynomial<937, 9> { using ZPZ = aerobus::zpz<937>; using type =
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<533>, ZPZV<483>, ZPZV<932»;
           }; // NOLINT
05081
                   template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
           POLYV<ZPZV<1>, ZPZV<939»; }; // NOLINT
                  template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
05082
           POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2»; }; // NOLINT
05083
                  template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2»; }; // NOLINT
05084
                  template<> struct ConwayPolynomial<941, 5> { using ZPZ = aerobus::zpz<941>; using type =
05085
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<939»; }; // NOLINT
05086
                  template<> struct ConwayPolynomial<941, 6> { using ZPZ = aerobus::zpz<941>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<459>, ZPZV<694>, ZPZV<538>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<941, 7> { using ZPZ = aerobus::zpz<941>; using type
05087
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<939»; }; //
                  template<> struct ConwayPolynomial<941, 8> { using ZPZ = aerobus::zpz<941>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<80>, ZPZV<675>, ZPZV<590>, ZPZV<590>, ZPZV<2»; }; //
05089
           template<> struct ConwayPolynomial<941, 9> { using ZPZ = aerobus::zpz<941>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<70>, ZPZV<708>, ZPZV<197>, ZPZV<939»;</pre>
           }; // NOLINT
                  template<> struct ConwayPolynomial<947, 1> { using ZPZ = aerobus::zpz<947>; using type =
           POLYV<ZPZV<1>, ZPZV<945»; }; // NOLINT
05091
                  template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
           POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
05092
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 4> { using ZPZ = aerobus::zpz<947>; using type =
05093
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2»; }; // NOLINT
05094
                  template<> struct ConwayPolynomial<947, 5> { using ZPZ = aerobus::zpz<947>; using type =
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 6> { using ZPZ = aerobus::zpz<947>; using type =
05095
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<880>, ZPZV<787>, ZPZV<95>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<947, 7> { using ZPZ = aerobus::zpz<947>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<945»; };
                  template<> struct ConwayPolynomial<947, 8> { using ZPZ = aerobus::zpz<947>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<845>, ZPZV<597>, ZPZV<581>, ZPZV<2»; }; //
           NOLINT
           template<> struct ConwayPolynomial<947, 9> { using ZPZ = aerobus::zpz<947>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<30>, ZPZV<30
05098
           }; // NOLINT
                  template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
05099
           POLYV<ZPZV<1>, ZPZV<950»; }; // NOLINT
0.5100
                  template<> struct ConwayPolynomial<953, 2> { using ZPZ = aerobus::zpz<953>; using type =
           POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
05101
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950»; }; // NOLINT template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
05102
           template<> struct ConwayPolynomial<953, 5> { using ZPZ = aerobus::zpz<953>; using type =
05103
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<950»; }; // NOLINT
                  template<> struct ConwayPolynomial<953, 6> { using ZPZ = aerobus::zpz<953>; using type =
05104
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<507>, ZPZV<829>, ZPZV<730>, ZPZV<3»; }; // NOLINI
                  template<> struct ConwayPolynomial<953, 7> { using ZPZ = aerobus::zpz<953>; using type
05105
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<550; // NOLINT template<> struct ConwayPolynomial<953, 8> { using ZPZ = aerobus::zpz<953>; using type =
05106
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6579>, ZPZV<658>, ZPZV<108>, ZPZV<3»; }; //
           NOLINT
05107
                  template<> struct ConwayPolynomial<953, 9> { using ZPZ = aerobus::zpz<953>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<819>, ZPZV<316>, ZPZV<950»;
           }; // NOLINT
05108
                  POLYV<ZPZV<1>, ZPZV<962»; }; // NOLINT
                  template<> struct ConwayPolynomial<967, 2> { using ZPZ = aerobus::zpz<967>; using type =
05109
```

174 File Documentation

```
POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5»; };
                   template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
0.5111
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<967, 5> { using ZPZ = aerobus::zpz<967>; using type =
05112
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<962»; }; // NOLINT
                   template<> struct ConwayPolynomial<967, 6> { using ZPZ = aerobus::zpz<967>; using type =
05113
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<948>, ZPZV<831>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<967, 7> { using ZPZ = aerobus::zpz<967>; using type =
05114
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<962»; }; // NOLINT
                 template<> struct ConwayPolynomial<967, 8> { using ZPZ = aerobus::zpz<967>; using type =
05115
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<840>, ZPZV<502>, ZPZV<136>, ZPZV<5»; }; //
05116
                 template<> struct ConwayPolynomial<967, 9> { using ZPZ = aerobus::zpz<967>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<512>, ZPZV<783>, ZPZV<962»;
           }; // NOLINT
                   template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
05117
           POLYV<ZPZV<1>, ZPZV<965»; }; // NOLINT
                  template<> struct ConwayPolynomial<971, 2> { using ZPZ = aerobus::zpz<971>; using type =
           POLYV<ZPZV<1>, ZPZV<970>, ZPZV<6»; }; // NOLINT
05119
                  template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<965»; }; // NOLINT template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<527>, ZPZV<6»; }; // NOLINT
05120
                   template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<965»; }; // NOLINT
05122
                  template<> struct ConwayPolynomial<971, 6> { using ZPZ = aerobus::zpz<971>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<970>, ZPZV<729>, ZPZV<718>, ZPZV<6»; }; // NOLINT
05123
                  template<> struct ConwayPolynomial<971, 7> { using ZPZ = aerobus::zpz<971>; using type =
          POLYVCZPZV<1>, ZPZV<0>, ZPZV<0
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<725>, ZPZV<281>, ZPZV<206>, ZPZV<6»; }; //
           template<> struct ConwayPolynomial<971, 9> { using ZPZ = aerobus::zpz<971>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<805>, ZPZV<473>, ZPZV<965»;</pre>
05125
           }; // NOLINT
                   template<> struct ConwayPolynomial<977, 1> { using ZPZ = aerobus::zpz<977>; using type =
           POLYV<ZPZV<1>, ZPZV<974»; }; // NOLINT
                  template<> struct ConwayPolynomial<977, 2> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<977, 3> { using ZPZ = aerobus::zpz<977>; using type =
05128
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<974»; }; // NOLINT template<> struct ConwayPolynomial<977, 4> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<800>, ZPZV<3»; }; // NOLINT
05130
                   template<> struct ConwayPolynomial<977, 5> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<974»; }; // NOLINT template<> struct ConwayPolynomial<977, 6> { using ZPZ = aerobus::zpz<977>; using type =
05131
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<729>, ZPZV<830>, ZPZV<753>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<977, 7> { using ZPZ = aerobus::zpz<977>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<974»; };
                  template<> struct ConwayPolynomial<977, 8> { using ZPZ = aerobus::zpz<977>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<855>, ZPZV<807>, ZPZV<77>, ZPZV<3»; }; //
                  template<> struct ConwayPolynomial<977, 9> { using ZPZ = aerobus::zpz<977>; using type =
05134
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<740>, ZPZV<974»;
           }; // NOLINT
   template<> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
05135
           POLYV<ZPZV<1>, ZPZV<978»; }; // NOLINT
                   template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
05136
           POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; }; // NOLINT
                  template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
05137
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978»; }; // NOLINT template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5»; }; // NOLINT
                  template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
05139
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<978»; }; // NOLINT
          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
05140
05141
                   template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<978»; }; // NOIIN template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
05142
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<5»; }; //
           NOLINT
                  template<> struct ConwayPolynomial<983, 9> { using ZPZ = aerobus::zpz<983>; using type
05143
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<9>, ZPZV<8>, ZPZV<8 , ZPZV<8
           }; // NOLINT
05144
                   template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
          POLYV<ZPZV<1>, ZPZV<985»; }; // NOLINT
                  template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
05145
           POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; }; // NOLINT
                   template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985»; }; // NOLINT template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
05147
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6»; }; // NOLINT 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
```

9.3 aerobus.h 175

```
template<> struct ConwayPolynomial<991, 6> { using ZPZ = aerobus::zpz<991>; using type =
              POLYV<ZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<637>, ZPZV<6575, ZPZV<278>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<991, 7> { using ZPZ = aerobus::zpz<991>; using type =
              POLYV-ZPZV-1>, ZPZV-0>, ZPZV-0
               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<9+, ZPZV<786>, ZPZV<234>, ZPZV<6*; }; //
05152
                         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<266>, ZPZV<285»;
               }; // NOLINT
05153
                        template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
              POLYV<ZPZV<1>, ZPZV<990»; }; // NOLINT
                         template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
05154
               POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; }; // NOLINT
05155
                       template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
             POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990»; ); // NOLINT
template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
05156
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<990»; }; // NOLINT
             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
             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<0>, ZPZV<0>; ZPZV<0>; ZPZV<0>; using ZPZ = aerobus::zpz<990»; }; // NOLINT template<> struct ConwayPolynomial<997, 8> { using ZPZ = aerobus::zpz<997>; using type =
05159
05160
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<934>, ZPZV<473>, ZPZV<241>, ZPZV<7»; }; //
              template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<732>, ZPZV<616>, ZPZV<990»;
05161
}; // NOLINT
05162 #endif // DO_NOT_DOCUMENT
05163 } // namespace aerobus
05164 #endif // AEROBUS_CONWAY_IMPORTS
05165
05166 #endif // __INC_AEROBUS__ // NOLINT
```

176 File Documentation

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 <i32::val<2>, i32::val<3>> addition operator yields v1 + v2 <i32::val<2>, i32::val<3>>

Template Parameters

v1	a value in i32
v2	a value in i32

10.5 i32::sub_t

substraction operator yields v1 - v2 <i32::val<3>, i32::val<2>> substraction operator yields v1 - v2 <i32::val<3>, i32::val<2>>

Template Parameters

v1	a value in i32
v2	a value in i32

10.6 i32::mul_t

multiplication operator yields v1 * v2 <i32::val<3>, i32::val<2>> multiplication operator yields v1 * v2 <i32::val<3>, i32::val<2>>

Template Parameters

v1	a value in i32
v2	a value in i32

10.7 i32::div_t

 $\label{eq:continuous} \mbox{division operator yields v1 / v2 < i32::val < 7>, i32::val < 2>> -> i32::val < 3> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7>, i32::val < 2>> -> i32::val < 3> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7>, i32::val < 2>> -> i32::val < 3> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7>, i32::val < 2>> -> i32::val < 3> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7>, i32::val < 7>, i32::val < 7> -> i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> -> i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> }$

v1	a value in i32
v2	a value in i32

10.11 i32::gcd_t 179

10.8 i32::gt_t

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

Template Parameters

v1	a value in i32
v2	a value in i32

10.9 i32::eq_t

$$\label{eq:constant} \begin{split} &\text{equality operator (type) yields v1 == v2 as std::integral_constant<bool> < i32::val<2>, i32::val<2>> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant<bool> < i32::val<2>, i32::val<2>> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant<bool> < i32::val<2>, i32::val<2>> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant<bool> < i32::val<2>, i32::val<2>> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std:$$

Template Parameters

v1	a value in i32
v2	a value in i32

10.10 i32::eq_v

equality operator (boolean value)

equality operator (boolean value)

Template Parameters

v1	
v2	<i32::val<1>, 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 < 6>, i32::val < 15>>

v1	a value in i32
v2	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 $\mathbf{v}>\mathbf{0}$ as boolean value

positivity (boolean value) yields $\mathbf{v}>\mathbf{0}$ as boolean value

Template Parameters

v a value in i32 <i32::val<1>>

10.14 i64::template

injects constant as an i64 value

injects constant as an i64 value

Template Parameters

x inject_constant_t<2>

10.15 i64::add_t

addition operator

addition operator

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

10.19 i64::mod_t 181

10.16 i64::sub_t

substraction operator

substraction operator

Template Parameters

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

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>></i64::val>

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>></i64::val>

10.19 i64::mod_t

modulus operator

modulus operator

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

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

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

10.21 i64::lt_t

Template Parameters

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

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

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

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

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

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

10.27 i64::pos_v 183

10.24 i64::eq_v

equality operator yields v1 == v2 as boolean value

equality operator yields v1 == v2 as boolean value

Template Parameters

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

10.25 i64::gcd_t

greatest common divisor yields GCD(v1, v2) as instanciation of i64::val

greatest common divisor yields GCD(v1, v2) as instanciation of i64::val

Template Parameters

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

10.26 i64::pos_t

is v posititive yields v>0 as std::true_type or std::false_type

is v posititive yields v > 0 as std::true_type or std::false_type

Template Parameters

v1 : an element of aerobus::i64::val <i64::val <1>>

10.27 i64::pos_v

positivity yields v > 0 as boolean value

positivity yields $\mathbf{v}>\mathbf{0}$ as boolean value

Template Parameters

v : an element of aerobus::i64::val <i64::val <1>>

10.28 polynomial

makes the constant (native type) polynomial a_0

makes the constant (native type) polynomial a_0

Template Parameters

x <i32>::template inject_constant_t<2>

10.29 q32::add_t

addition operator

addition operator

Template Parameters

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

10.30 FractionField

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

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

Template Parameters

Ring <i64> is q64 (rationals with 64 bits numerator and denominator)

10.31 aerobus::ContinuedFraction

represents a continued fraction a0 + $\frac{1}{a_1 + \frac{1}{a_2 + \dots}}$

represents a continued fraction a0 + $\frac{1}{a_1 + \frac{1}{a_2 + \dots}}$ [https://en.wikipedia.org/wiki/Continued_ \leftarrow fraction](See in Wikipedia)

values	are
	int64_t

10.32 Pl_fraction::val

<1, 1, 1> represents
$$1+\frac{1}{\frac{1}{1}}$$

10.32 Pl_fraction::val

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

10.33 E_fraction::val

approximation of e -> 2.718...

approximation of $e \rightarrow 2.718...$

Index

```
abs t
                                                               sin, 28
                                                               sinh, 28
     aerobus, 21
                                                               SQRT2 fraction, 28
add t
     aerobus::i32, 48
                                                               SQRT3 fraction, 29
     aerobus::i64, 51
                                                               stirling_signed_t, 29
     aerobus::polynomial < Ring >, 57
                                                               stirling_unsigned_t, 29
     aerobus::Quotient < Ring, X >, 64
                                                               tan, 29
     aerobus::zpz, 86
                                                               tanh, 30
addfractions_t
                                                               taylor, 30
     aerobus, 21
                                                               vadd t, 30
aerobus, 17
                                                               vmul t, 30
     abs t, 21
                                                          aerobus::ContinuedFraction < a0 >, 44
     addfractions_t, 21
                                                               type, 45
     aligned_malloc, 31
                                                               val, 45
     alternate t, 21
                                                          aerobus::ContinuedFraction < a0, rest... >, 45
     alternate_v, 31
                                                               type, 46
                                                               val, 46
     asin, 21
     asinh, 22
                                                          aerobus::ContinuedFraction< values >, 44
     atan, 22
                                                          aerobus::ConwayPolynomial, 46
     atanh, 22
                                                          aerobus::i32, 46
    bell t, 22
                                                               add t, 48
    bernoulli t, 23
                                                               div t, 48
    bernoulli v, 32
                                                               eq t, 48
    combination_t, 23
                                                               eq_v, 49
    combination_v, 32
                                                               gcd_t, 48
     cos, 23
                                                               gt t, 48
    cosh, 24
                                                               inject_constant_t, 48
                                                               inject_ring_t, 48
     E_fraction, 24
     exp, 24
                                                               inner_type, 48
     expm1, 24
                                                               is_euclidean_domain, 49
     factorial t, 24
                                                               is field, 49
    factorial_v, 32
                                                               It_t, 48
     field, 31
                                                               mod t, 48
     fpq32, 25
                                                               mul t, 49
     fpq64, 25
                                                               one, 49
     FractionField, 25
                                                               pos_t, 49
    gcd_t, 25
                                                               pos_v, 50
    geometric_sum, 25
                                                               sub t, 49
    Inp1, 26
                                                               zero, 49
     make_q32_t, 26
                                                          aerobus::i32::val< x >, 72
     make_q64_t, 26
                                                               enclosing_type, 73
     makefraction t, 26
                                                               eval, 73
     mulfractions t, 27
                                                               get, 73
     pi64, 27
                                                               is_zero_t, 73
     PI fraction, 27
                                                               to string, 74
                                                               v, 74
     pow t, 27
    pq64, 27
                                                          aerobus::i64, 50
     q32, 28
                                                               add_t, 51
     q64, 28
                                                               div_t, 51
```

eq_t, 51	is_euclidean_domain, 61
eq_v, 53	is_field, 61
gcd_t, 51	lt_t, 58
gt_t, 52	mod_t, 59
gt_v, 53	monomial_t, 59
inject_constant_t, 52	mul_t, 59
inject_ring_t, 52	one, 60
inner_type, 52	pos_t, 60
is_euclidean_domain, 54	pos_v, 61
is_field, 54	simplify_t, 60
lt_t, 52	sub_t, 60
It_v, 54	X, 61
mod_t, 52	zero, 61
mul_t, 52	aerobus::polynomial< Ring >::val< coeffN >, 82
one, 53	aN, 83
pos_t, 53 pos_v, 54	coeff_at_t, 83
sub_t, 53	degree, 84 enclosing_type, 83
zero, 53	enclosing_type, 83 eval, 83
aerobus::i64::val< x >, 74	is_zero_t, 83
enclosing_type, 75	is_zero_v, 84
eval, 75	strip, 83
get, 75	to_string, 83
inner_type, 75	aerobus::polynomial< Ring >::val< coeffN >::coeff_at<
is_zero_t, 75	index, E >, 43
to_string, 76	aerobus::polynomial< Ring >::val< coeffN >::coeff_at<
v, 76	index, std::enable_if_t<(index< 0 index >
aerobus::internal, 33	0)>>, 43
index_sequence_reverse, 36	type, 43
is instantiation of v. 36	aerobus::polynomial < Ring >::val < coeffN >::coeff at <
is_instantiation_of_v, 36 make index sequence reverse, 36	aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable if t < (index==0) > >, 44
make_index_sequence_reverse, 36	index, std::enable_if_t<(index==0)>>, 44
make_index_sequence_reverse, 36 type_at_t, 36	index, std::enable_if_t<(index==0)>>, 44 type, 44
make_index_sequence_reverse, 36	index, std::enable_if_t<(index==0)>>, 44
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime< n >, 54	index, std::enable_if_t<(index==0)>>, $\frac{44}{4}$ type, $\frac{44}{4}$ aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >, 76
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime< n >, 54 value, 55 aerobus::IsEuclideanDomain, 41	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >, 76 aN, 77
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime< n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >, 76 aN, 77 coeff_at_t, 77
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >, 76 aN, 77 coeff_at_t, 77 degree, 79
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >, 76 aN, 77 coeff_at_t, 77 degree, 79 enclosing_type, 78
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime< n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::lsEuclideanDomain, 41 aerobus::lsField, 41 aerobus::knig, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40 physicist, 40	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40 physicist, 40 probabilist, 40	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40 physicist, 40 aerobus::polynomial < Ring >, 55	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::lsEuclideanDomain, 41 aerobus::lsField, 41 aerobus::lsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40 physicist, 40 aerobus::polynomial < Ring >, 55 add_t, 57	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40 physicist, 40 probabilist, 40 aerobus::polynomial < Ring >, 55 add_t, 57 derive_t, 57	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40 physicist, 40 probabilist, 40 aerobus::polynomial < Ring >, 55 add_t, 57 derive_t, 57 div_t, 57	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40 physicist, 40 probabilist, 40 aerobus::polynomial < Ring >, 55 add_t, 57 derive_t, 57 div_t, 57 eq_t, 57	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40 physicist, 40 probabilist, 40 aerobus::polynomial < Ring >, 55 add_t, 57 derive_t, 57 div_t, 57 eq_t, 57 gcd_t, 58	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40 physicist, 40 probabilist, 40 aerobus::polynomial < Ring >, 55 add_t, 57 derive_t, 57 div_t, 57 eq_t, 57 gcd_t, 58 gt_t, 58	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40 physicist, 40 probabilist, 40 aerobus::polynomial < Ring >, 55 add_t, 57 derive_t, 57 div_t, 57 eq_t, 57 gcd_t, 58 gt_t, 58 inject_constant_t, 58	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime < n >, 54 value, 55 aerobus::IsEuclideanDomain, 41 aerobus::IsField, 41 aerobus::IsRing, 42 aerobus::known_polynomials, 36 bernoulli, 37 bernstein, 38 chebyshev_T, 38 chebyshev_U, 38 hermite_kind, 40 hermite_phys, 39 hermite_prob, 39 laguerre, 39 legendre, 40 physicist, 40 probabilist, 40 aerobus::polynomial < Ring >, 55 add_t, 57 derive_t, 57 div_t, 57 eq_t, 57 gcd_t, 58 gt_t, 58	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >,

type, 80	aerobus::polynomial< Ring >::val< coeffN, coeffs
aerobus::type_list< Ts >, 68	>, 77
at, 69	asin
concat, 69	aerobus, 21
insert, 69	asinh
length, 70	aerobus, 22
push_back, 69	at
push_front, 70	aerobus::type_list< Ts >, 69
remove, 70	atan
aerobus::type_list< Ts >::pop_front, 61	aerobus, 22
tail, 62	atanh
type, 62	aerobus, 22
aerobus::type_list< Ts >::split< index >, 67	30.000,
head, 67	bell_t
tail, 67	aerobus, 22
aerobus::type_list<>, 70	bernoulli
concat, 71	aerobus::known_polynomials, 37
insert, 71	bernoulli_t
length, 72	aerobus, 23
push_back, 71	bernoulli_v
push_front, 71	aerobus, 32
aerobus::zpz $<$ p $>$, 84	bernstein
add_t, 86	aerobus::known polynomials, 38
div_t, 86	
eq_t, 86	chebyshev_T
eq_v, 89	aerobus::known_polynomials, 38
gcd_t, 87	chebyshev_U
gt_t, 87	aerobus::known_polynomials, 38
gt_v, 89	coeff_at_t
	aerobus::polynomial< Ring >::val< coeffN >, 83
inject_constant_t, 87 inner_type, 87	aerobus::polynomial< Ring >::val< coeffN, coeffs
is_euclidean_domain, 90	>, 77
is field, 90	combination_t
It_t, 87	aerobus, 23
	combination_v
It_v, 90	aerobus, 32
mod_t, 88	concat
mul_t, 88 one, 88	aerobus::type_list< Ts >, 69
	aerobus::type_list<>, 71
pos_t, 88	cos
pos_v, 90	aerobus, 23
sub_t, 89 zero, 89	cosh
	aerobus, 24
aerobus:: $zpz ::val < x >, 80$,
enclosing_type, 81	degree
eval, 81	aerobus::polynomial< Ring >::val< coeffN >, 84
get, 81	aerobus::polynomial< Ring >::val< coeffN, coeffs
is_zero_t, 81	>, 79
to_string, 81	derive_t
v, 82	aerobus::polynomial $<$ Ring $>$, 57
aligned_malloc	div_t
aerobus, 31	aerobus::i32, 48
alternate_t	aerobus::i64, 51
aerobus, 21	aerobus::polynomial < Ring >, 57
alternate_v	aerobus::Quotient< Ring, X >, 64
aerobus, 31	aerobus:: $zpz $, 86
aN (N)	• • •
aerobus::polynomial $<$ Ring $>$::val $<$ coeffN $>$, 83	E_fraction
	aerobus, 24

enclosing_type	aerobus::zpz, 87
aerobus::i32::val $< x >$, 73	gt_v
aerobus::i64::val $<$ x $>$, 75	aerobus::i64, 53
aerobus::polynomial $<$ Ring $>$::val $<$ coeffN $>$, 83	aerobus:: $zpz $, 89
aerobus::polynomial< Ring >::val< coeffN, coeffs	
>, 78	head
aerobus::zpz $<$ p $>$::val $<$ x $>$, 81	aerobus::type_list< Ts >::split< index >, 67
eq_t	hermite_kind
aerobus::i32, 48	aerobus::known_polynomials, 40
aerobus::i64, 51	hermite_phys
aerobus::polynomial < Ring >, 57	aerobus::known_polynomials, 39
aerobus::Quotient< Ring, X >, 64	hermite_prob
aerobus::zpz, 86	aerobus::known_polynomials, 39
eq_v	index_sequence_reverse
aerobus::i32, 49	aerobus::internal, 36
aerobus::i64, 53	inject_constant_t
aerobus::Quotient< Ring, X >, 66	aerobus::i32, 48
aerobus::zpz, 89	aerobus::i64, 52
eval	aerobus::polynomial< Ring >, 58
aerobus::i32::val $< x >$, 73	aerobus::Quotient< Ring, X >, 64
aerobus::i64::val $<$ x $>$, 75	aerobus::zpz, 87
aerobus::polynomial< Ring >::val< coeffN >, 83	inject_ring_t
aerobus::polynomial< Ring >::val< coeffN, coeffs	aerobus::i32, 48
>, 78	aerobus::i64, 52
aerobus:: $zpz $:: $val < x >$, 81	aerobus::io4, 32 aerobus::polynomial< Ring >, 58
exp	aerobus::Quotient< Ring, X >, 65
aerobus, 24	
expm1	inner_type
aerobus, 24	aerobus::i32, 48
	aerobus::i64, 52
factorial_t	aerobus::i64::val< x >, 75
aerobus, 24	aerobus::zpz, 87
factorial_v	insert Search continue list (Ta > CO
aerobus, 32	aerobus::type_list< Ts >, 69
field	aerobus::type_list<>>, 71
aerobus, 31	Introduction, 1
fpq32	is_euclidean_domain
aerobus, 25	aerobus::i32, 49
fpq64	aerobus::i64, 54
aerobus, 25	aerobus::polynomial < Ring >, 61
FractionField	aerobus::Quotient< Ring, X >, 66
aerobus, 25	aerobus:: $zpz $, 90
and t	is_field
gcd_t	aerobus::i32, 49
aerobus, 25	aerobus::i64, 54
aerobus::i32, 48	aerobus::polynomial< Ring >, 61
aerobus::i64, 51	aerobus::zpz, 90
aerobus::polynomial < Ring >, 58	is_instantiation_of_v
aerobus::zpz, 87	aerobus::internal, 36
geometric_sum	is_zero_t
aerobus, 25	aerobus::i32::val< x >, 73
get	aerobus::i64::val < x >, 75
aerobus::i32::val $< x >$, 73	aerobus::polynomial < Ring >::val < coeffN >, 83
aerobus::i64::val< x >, 75	aerobus::polynomial< Ring >::val< coeffN, coeffs
aerobus::zpz::val< x >, 81	>, 78
gt_t	aerobus::zpz $<$ p $>::val<x>, 81$
aerobus::i32, 48	is_zero_v
aerobus::i64, 52	aerobus::polynomial< Ring >::val< coeffN >, 84
aerobus::polynomial < Ring >, 58	

aerobus::polynomial< Ring >::val< coeffN, coeffs >, 79	pos_t aerobus::i32, 49
loguage	aerobus::i64, 53
laguerre	aerobus::polynomial < Ring >, 60
aerobus::known_polynomials, 39	aerobus::Quotient < Ring, X >, 65
legendre	aerobus::zpz, 88
aerobus::known_polynomials, 40	pos_v
length aerobus::type_list< Ts >, 70	aerobus::i32, 50
aerobus::type_list<>>, 72	aerobus::i64, 54
Inp1	aerobus::polynomial < Ring >, 61
•	aerobus::Quotient < Ring, X >, 66
aerobus, 26	aerobus::zpz, 90
It_t	pow_t
aerobus::i32, 48	aerobus, 27
aerobus::i64, 52	pq64
aerobus::polynomial < Ring >, 58	aerobus, 27
aerobus::zpz, 87	probabilist
It_v	aerobus::known_polynomials, 40
aerobus::i64, 54	push_back
aerobus::zpz, 90	aerobus::type_list< Ts >, 69
made index agains as various	aerobus::type_list<>, 71
make_index_sequence_reverse	push_front
aerobus::internal, 36	aerobus::type_list< Ts >, 70
make_q32_t	aerobus::type_list<>, 71
aerobus, 26	
make_q64_t	q32
aerobus, 26	aerobus, 28
makefraction_t	q64
aerobus, 26	aerobus, 28
mod_t	
aerobus::i32, 48	README.md, 91
aerobus::i64, 52	remove
aerobus::polynomial< Ring >, 59	aerobus::type_list< Ts >, 70
aerobus::Quotient< Ring, X >, 65	almostific a
aerobus:: $zpz $, 88	simplify_t
monomial_t	aerobus::polynomial< Ring >, 60
aerobus::polynomial< Ring >, 59	sin
mul_t	aerobus, 28
aerobus::i32, 49	sinh
aerobus::i64, 52	aerobus, 28
aerobus::polynomial $<$ Ring $>$, 59	SQRT2_fraction
aerobus::Quotient $<$ Ring, $X >$, 65	aerobus, 28
aerobus::zpz, 88	SQRT3_fraction
mulfractions_t	aerobus, 29
aerobus, 27	src/aerobus.h, 91
	stirling_signed_t
one	aerobus, 29
aerobus::i32, 49	stirling_unsigned_t
aerobus::i64, 53	aerobus, 29
aerobus::polynomial $<$ Ring $>$, 60	strip
aerobus::Quotient< Ring, X >, 65	aerobus::polynomial< Ring >::val< coeffN >, 83
aerobus::zpz, 88	aerobus::polynomial< Ring >::val< coeffN, coeffs
	>, 78
physicist	sub_t
aerobus::known_polynomials, 40	aerobus::i32, 49
pi64	aerobus::i64, 53
aerobus, 27	aerobus::polynomial< Ring >, 60
PI_fraction	aerobus::zpz, 89
aerobus, 27	

```
tail
     aerobus::type_list< Ts >::pop_front, 62
     aerobus::type_list< Ts >::split< index >, 67
tan
     aerobus, 29
tanh
     aerobus, 30
taylor
     aerobus, 30
to_string
    aerobus::i32::val< x >, 74
    aerobus::i64::val< x >, 76
     aerobus::polynomial < Ring >::val < coeffN >, 83
     aerobus::polynomial< Ring >::val< coeffN, coeffs
          >, 79
     aerobus::zpz ::val < x >, 81
type
     aerobus::ContinuedFraction < a0 >, 45
     aerobus::ContinuedFraction< a0, rest... >, 46
     aerobus::polynomial< Ring >::val< coeffN
          >::coeff_at< index, std::enable_if_t<(index<
         0 \mid | \text{index} > 0) > , 43
     aerobus::polynomial< Ring
                                   >::val<
                                               coeffN
          >::coeff_at< index, std::enable_if_t<(index==0)>
          >, 44
     aerobus::Quotient< Ring, X >::val< V >, 80
    aerobus::type_list< Ts >::pop_front, 62
type at t
    aerobus::internal, 36
٧
     aerobus::i32::val < x >, 74
     aerobus::i64::val< x >, 76
    aerobus::zpz<p>::val<math><x>, 82
vadd t
     aerobus, 30
val
    aerobus::ContinuedFraction< a0 >, 45
     aerobus::ContinuedFraction< a0, rest... >, 46
value
     aerobus::is_prime< n >, 55
vmul t
     aerobus, 30
Χ
     aerobus::polynomial < Ring >, 61
zero
     aerobus::i32, 49
    aerobus::i64, 53
     aerobus::polynomial < Ring >, 61
     aerobus::Quotient < Ring, X >, 66
     aerobus::zpz< p>, 89
```