# 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	40
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	43
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	44
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	75
8.18.2 Member Typedef Documentation	75
8.18.2.1 enclosing_type	75
8.18.2.2 inner_type	75
8.18.2.3 is_zero_t	75
8.18.3 Member Function Documentation	75
8.18.3.1 eval()	75
8.18.3.2 get()	76
8.18.3.3 to_string()	76
8.18.4 Member Data Documentation	76
8.18.4.1 v	76
8.19 aerobus::polynomial < Ring >::val < coeffN, coeffs > Struct Template Reference	76
8.19.1 Detailed Description	77
8.19.2 Member Typedef Documentation	77
8.19.2.1 aN	77
8.19.2.2 coeff_at_t	78
8.19.2.3 enclosing_type	78
8.19.2.4 is_zero_t	78
8.19.2.5 strip	78
8.19.3 Member Function Documentation	78

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

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

Inde	ndex 189		
-	10.33 E_fraction::val	187	
	10.32 PI_fraction::val		
•	10.31 aerobus::ContinuedFraction	186	
-	10.30 FractionField	186	
-	10.29 q32::add_t	186	
-	10.28 polynomial	186	
-	10.27 i64::pos_v	185	
-	10.26 i64::pos_t	185	
-	10.25 i64::gcd_t	185	
-	10.24 i64::eq_v	185	
-	10.23 i64::eq_t	184	
-	10.22 i64::lt_v	184	

# 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  $\pi$ , 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	80

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 brief descriptions:	
src/aerobus.h	93

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</li>

#### **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

## **Template Parameters**

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

## **Template Parameters**

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)<sup>^</sup>k as value from T

#### **Template Parameters**

```
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, I >

struct pop\_front\_h

```
• struct bernstein_helper< i, m, I, std::enable_if_t<(m > 0) &&(i > 0) &&(i < m)> >

    struct bernstein_helper< i, m, I, std::enable_if_t<(m > 0) &&(i==0)> >

    struct bernstein_helper< i, m, I, std::enable_if_t<(m > 0) &&(i==m)> >

    struct chebyshev_helper

    struct chebyshev helper< 1, 0, I >

    struct chebyshev_helper< 1, 1, I >

    struct chebyshev_helper< 2, 0, I >

    struct chebyshev_helper< 2, 1, I >

· struct combination

    struct combination helper

    struct combination_helper< T, 0, n >

    struct combination_helper< T, k, n, std::enable_if_t<(n >=0 &&k >(n/2) &&k > 0)> >

    struct combination_helper< T, k, n, std::enable_if_t<(n >=0 &&k<=(n/2) &&k > 0)> >

    struct cos_coeff

· struct cos coeff helper

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

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

    struct cosh coeff

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

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

    struct exp_coeff

    struct factorial

    struct factorial < T, 0 >

struct factorial < T, x, std::enable_if_t < (x > 0) > >

    struct FractionFieldImpl

    struct FractionFieldImpl< Field, std::enable if t< Field::is field >>

    struct FractionFieldImpl< Ring, std::enable_if_t<!Ring::is_field >>

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

    struct gcd< Ring, std::enable_if_t< Ring::is_euclidean_domain > >

    struct geom_coeff

· struct hermite helper

    struct hermite_helper< 0, known_polynomials::hermite_kind::physicist, I >

    struct hermite_helper< 0, known_polynomials::hermite_kind::probabilist, I >

    struct hermite helper< 1, known polynomials::hermite kind::physicist, I >

    struct hermite_helper< 1, known_polynomials::hermite_kind::probabilist, I >

    struct hermite helper< deg, known polynomials::hermite kind::physicist, I >

    struct hermite_helper< deg, known_polynomials::hermite_kind::probabilist, l >

• struct insert h
· struct is instantiation of

    struct is_instantiation_of< TT, TT< Ts... >>

    struct laguerre helper

    struct laguerre_helper< 0, I >

    struct laguerre_helper< 1, I >

· struct legendre helper

    struct legendre helper< 0, I >

    struct legendre_helper< 1, I >

    struct 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, typename I = aerobus::i64>
  using chebyshev_T = typename internal::chebyshev_helper< 1, deg, I >::type
      Chebyshev polynomials of first kind.
• template < size_t deg, typename I = aerobus::i64>
  using chebyshev U = typename internal::chebyshev helper< 2, deg, I >::type
      Chebyshev polynomials of second kind.
• template<size_t deg, typename I = aerobus::i64>
  using laguerre = typename internal::laguerre_helper< deg, l >::type
     Laguerre polynomials.
• template<size_t deg, typename I = aerobus::i64>
  using hermite_prob = typename internal::hermite_helper< deg, hermite_kind::probabilist, I >::type
     Hermite polynomials - probabilist form.
• template < size_t deg, typename I = aerobus::i64>
  using hermite_phys = typename internal::hermite_helper< deg, hermite_kind::physicist, I >::type
      Hermite polynomials - physicist form.
• template<size_t i, size_t m, typename I = aerobus::i64>
  using bernstein = typename internal::bernstein_helper< i, m, I >::type
     Bernstein polynomials.
• template < size t deg, typename I = aerobus::i64>
  using legendre = typename internal::legendre_helper< deg, I >::type
     Legendre polynomials.
• template < size_t deg, typename I = aerobus::i64>
  using bernoulli = taylor< I, internal::bernoulli_coeff< deg >::template inner, deg >
     Bernoulli polynomials.
```

#### **Enumerations**

enum hermite\_kind { probabilist , physicist }

## 6.3.1 Detailed Description

families of well known polynomials such as Hermite or Bernstein

## 6.3.2 Typedef Documentation

## 6.3.2.1 bernoulli

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::bernoulli = typedef taylor<I, internal::bernoulli_coeff<deg>←
::template inner, deg>
```

Bernoulli polynomials.

Lives in polynomial<FractionField<I>>

#### See also

```
See in Wikipedia
```

## **Template Parameters**

deg	degree of polynomial
1	Integers ring (defaults to aerobus::i64)

#### 6.3.2.2 bernstein

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

Bernstein polynomials.

Lives in polynomial

See also

See in Wikipedia

#### **Template Parameters**

i	index of polynomial (between 0 and m)
m	degree of polynomial
I	Integers ring (defaults to aerobus::i64)

#### 6.3.2.3 chebyshev\_T

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

Chebyshev polynomials of first kind.

See also

```
See in Wikipedia
```

#### **Template Parameters**

deg	degree of polynomial
integer	rings (defaults to aerobus::i64)

## 6.3.2.4 chebyshev\_U

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

using aerobus::known\_polynomials::chebyshev\_U = typedef typename internal::chebyshev\_helper<2,
deg, I>::type

Chebyshev polynomials of second kind.

Lives in polynomial

See also

```
See in Wikipedia
```

#### **Template Parameters**

deg	degree of polynomial
integer	rings (defaults to aerobus::i64)

#### 6.3.2.5 hermite\_phys

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

Hermite polynomials - physicist form.

See also

```
See in Wikipedia
```

#### **Template Parameters**

```
deg degree of polynomial
```

#### 6.3.2.6 hermite\_prob

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

Hermite polynomials - probabilist form.

See also

```
See in Wikipedia
```

## **Template Parameters**

deg	degree of polynomial

#### 6.3.2.7 laguerre

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

Laguerre polynomials.

Lives in polynomial<FractionField<I>>

See also

```
See in Wikipedia
```

#### **Template Parameters**

deg	degree of polynomial
1	Integers ring (defaults to aerobus::i64)

#### 6.3.2.8 legendre

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

Legendre polynomials.

Lives in polynomial<FractionField<I>>

See also

```
See in Wikipedia
```

## **Template Parameters**

deg	degree of polynomial
1	Integers Ring (defaults to aerobus::i64)

## **6.3.3 Enumeration Type Documentation**

## 6.3.3.1 hermite\_kind

enum aerobus::known\_polynomials::hermite\_kind

#### Enumerator

probabilist	
physicist	

# **Chapter 7**

# **Concept Documentation**

## 7.1 aerobus::IsEuclideanDomain Concept Reference

Concept to express R is an euclidean domain.

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

#### 7.1.1 Concept definition

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

#### **Template Parameters**

```
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>>

#### **Template Parameters**

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

**Template Parameters** 

```
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 INLINED DEVICE valueType get ()
```

cast x into valueType

• static std::string to\_string ()

string representation of value

template<typename valueRing >

static constexpr DEVICE INLINED valueRing eval (const valueRing &v)

cast x into valueRing

#### **Static Public Attributes**

static constexpr int32\_t v = x
 actual value stored in val type

## 8.17.1 Detailed Description

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

values in i32, again represented as types

**Template Parameters** 

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

cast x into valueType

## **Template Parameters**

```
valueType double for example
```

## 8.17.3.3 to\_string()

```
\label{template} $$ \text{template} < \text{int32\_t x} > $$ \text{static std::string aerobus::i32::val} < $$ x > ::to\_string () [inline], [static] $$ $$ $$ \text{template} < \text{template
```

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 DEVICE INLINED valueType get ()
```

cast value in valueType

• static std::string to\_string ()

string representation

template<typename valueRing >
 static constexpr DEVICE INLINED valueRing eval (const valueRing &v)

cast value in valueRing

#### **Static Public Attributes**

static constexpr int64\_t v = x
 actual value

## 8.18.1 Detailed Description

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

values in i64

Template Parameters

x an actual integer
```

## 8.18.2 Member Typedef Documentation

#### 8.18.2.1 enclosing\_type

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

## 8.18.2.2 inner\_type

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

type of represented values

#### 8.18.2.3 is\_zero\_t

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

## 8.18.3 Member Function Documentation

## 8.18.3.1 eval()

## **Template Parameters**

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

#### 8.18.3.2 get()

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

cast value in valueType

**Template Parameters** 

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

## 8.18.3.3 to\_string()

string representation

```
\label{template} $$ \template< int64_t x> $$ 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 DEVICE INLINED valueRing eval (const valueRing &x)

evaluates polynomial seen as a function operating on ValueRing

#### **Static Public Attributes**

static constexpr size\_t degree = sizeof...(coeffs)
 degree of the polynomial
 static constexpr bool is\_zero\_v = is\_zero\_t::value

true if polynomial is constant zero

## 8.19.1 Detailed Description

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

values (seen as types) in polynomial ring

Template Parameters
```

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

## **Static Public Attributes**

static constexpr int32\_t v = x % p
 actual value

## 8.21.1 Member Typedef Documentation

#### 8.21.1.1 enclosing type

```
template<int32_t p>
template<int32_t x>
using aerobus::zpz::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<iint32_t x>
template<typename valueType >
static constexpr DEVICE INLINED 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 DEVICE INLINED valueRing eval (const valueRing &x)

#### **Static Public Attributes**

- static constexpr size\_t degree = 0
   degree
- static constexpr bool is\_zero\_v = is\_zero\_t::value

## 8.22.1 Detailed Description

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

specialization for constants

Template Parameters

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]
```

# 8.22.4.2 is zero v

degree

```
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**

```
    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<iint32_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<iint32_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<int32_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<iint32_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

```
{\tt template}{<} {\tt int32\_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

```
\label{lem:lambda} $$ \end{template} $$$ \end{
```

equality operator (booleanvalue)

## **Template Parameters**

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

## 8.23.3.2 gt\_v

```
{\tt template}{<} {\tt int32\_t} \ p{>}
```

## 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 It 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
00019 #ifdef _MSC_VER
00020 #define ALIGNED(x) __declspec(align(x))
00021 #define INLINED __forceinline
00023 #define ALIGNED(x) __attribute__((aligned(x)))
00024 #define INLINED __attribute__((always_inline)) inline
00025 #endif
00026
00027 #ifdef __CUDACC_
00028 #define DEVICE __host__ _device__
```

94 File Documentation

```
00029 #else
00030 #define DEVICE
00031 #endif
00032
00034
00036
00038
00039 // aligned allocation
00040 namespace aerobus {
00047
           template<typename T>
           T* aligned_malloc(size_t count, size_t alignment) {
00048
               #ifdef _MSC_VER
00049
00050
               return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
00051
00052
               return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
00053
               #endif
00054
00055 } // namespace aerobus
00056
00057 // concepts
00058 namespace aerobus {
00060
          template <typename R>
00061
           concept IsRing = requires {
00062
               typename R::one;
00063
               typename R::zero;
00064
               typename R::template add_t<typename R::one, typename R::one>;
00065
               typename R::template sub_t<typename R::one, typename R::one>;
00066
               typename R::template mul_t<typename R::one, typename R::one>;
00067
00068
00070
           template <typename R>
00071
           concept IsEuclideanDomain = IsRing<R> && requires {
00072
               typename R::template div_t<typename R::one, typename R::one>;
00073
               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>;
00074
00075
00076
00078
               R::template pos_v<typename R::one> == true;
00079
                // typename R::template gt_t<typename R::one, typename R::zero>;
00080
               R::is_euclidean_domain == true;
00081
          };
00082
00084
           template<typename R>
           concept IsField = IsEuclideanDomain<R> && requires {
00085
00086
              R::is_field == true;
00087
00088 } // namespace aerobus
00089
00090 // utilities
00091 namespace aerobus {
00092
           namespace internal {
00093
               template<template<typename...> typename TT, typename T>
00094
               struct is_instantiation_of : std::false_type { };
00095
00096
               template<template<typename...> typename TT, typename... Ts>
struct is_instantiation_of<TT, TT<Ts...» : std::true_type { };</pre>
00097
00098
00099
               template<template<typename...> typename TT, typename T>
00100
               inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00101
00102
               template <int64_t i, typename T, typename... Ts>
00103
               struct type_at {
                   static_assert(i < sizeof...(Ts) + 1, "index out of range");
using type = typename type_at<i - 1, Ts...>::type;
00104
00105
00106
00107
00108
               template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00109
                   using type = T:
00110
00111
00112
               template <size_t i, typename... Ts>
00113
               using type_at_t = typename type_at<i, Ts...>::type;
00114
00115
00116
               template<size_t n, size_t i, typename E = void>
00117
               struct _is_prime {};
00118
00119
               template<size t i>
               struct _{is\_prime<0, i> \{}
00120
                   static constexpr bool value = false;
00121
00122
00123
00124
               template<size_t i>
00125
               struct _is_prime<1, i> {
                    static constexpr bool value = false;
00126
00127
               };
```

9.3 aerobus.h

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

96 File Documentation

```
template<typename Ring, typename E = void>
00227
00228
00229
              template<typename Ring>
00230
              struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
00231
                  template<typename A, typename B, typename E = void>
                  struct gcd_helper {};
00232
00233
00234
                  // B = 0, A > 0
                  00235
00236
00237
00238
                          (Ring::template gt_t<A, typename Ring::zero>::value))» {
00239
                      using type = A;
00240
                  };
00241
                  // B = 0, A < 0
00242
00243
                  template<typename A, typename B>
                  struct gcd_helper<A, B, std::enable_if_t<
00244
00245
                      ((B::is_zero_t::value) &&
00246
                          !(Ring::template gt_t<A, typename Ring::zero>::value))» {
00247
                      using type = typename Ring::template sub_t<typename Ring::zero, A>;
00248
                  };
00249
00250
                  // B != 0
                  template<typename A, typename B>
struct gcd_helper<A, B, std::enable_if_t<</pre>
00251
00252
00253
                      (!B::is_zero_t::value)
00254
                  private: // NOLINT
00255
00256
                      // A / B
00257
                      using k = typename Ring::template div_t<A, B>;
00258
                      // A - (A/B) *B = A % B
00259
                      using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
00260
00261
                  public:
00262
                      using type = typename gcd_helper<B, m>::type;
00263
00264
00265
                  template<typename A, typename B>
00266
                  using type = typename gcd_helper<A, B>::type;
00267
              };
00268
         } // namespace internal
00269
          // vadd and vmul
00270
00271
          namespace internal {
00272
             template<typename... vals>
00273
              struct vmul {};
00274
00275
              template<typename v1, typename... vals>
00276
              struct vmul<v1, vals...> {
                 using type = typename v1::enclosing_type::template mul_t<v1, typename
00277
     vmul<vals...>::type>;
00278
             };
00279
00280
              template<typename v1>
              struct vmul<v1> {
00281
00282
                 using type = v1;
00283
00284
00285
              template<typename... vals>
00286
             struct vadd {};
00287
00288
              template<typename v1, typename... vals>
00289
              struct vadd<v1, vals...> {
00290
                using type = typename v1::enclosing_type::template add_t<v1, typename
     vadd<vals...>::type>;
00291
             };
00292
00293
              template<typename v1>
00294
              struct vadd<v1> {
00295
                  using type = v1;
00296
00297
          } // namespace internal
00298
00301
          template<typename T, typename A, typename B>
00302
          using gcd_t = typename internal::gcd<T>::template type<A, B>;
00303
00307
          template<typename... vals>
00308
          using vadd_t = typename internal::vadd<vals...>::type;
00309
00313
          template<typename... vals>
00314
          using vmul_t = typename internal::vmul<vals...>::type;
00315
00319
          template<typename val>
          requires IsEuclideanDomain<typename val::enclosing_type>
00320
00321
          using abs t = std::conditional t<
```

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

```
using s = typename L::template split<index>;
                   using left = typename s::head;
using right = typename s::tail;
00453
00454
                   using ll = typename left::template push_back<T>;
00455
00456
                   using type = typename ll::template concat<right>;
00457
               };
00458
00459
               template <size_t index, typename L>
00460
               struct remove_h {
                   using s = typename L::template split<index>;
using left = typename s::head;
using right = typename s::tail;
00461
00462
00463
00464
                   using rr = typename right::pop_front::tail;
00465
                   using type = typename left::template concat<rr>;
00466
00467
          } // namespace internal
00468
00472
          template <typename... Ts>
00473
          struct type_list {
00474
           private:
00475
              template <typename T>
00476
               struct concat_h;
00477
00478
               template <typename... Us>
struct concat_h<type_list<Us...» {</pre>
00479
00480
                  using type = type_list<Ts..., Us...>;
00481
00482
00483
            public:
00485
               static constexpr size_t length = sizeof...(Ts);
00486
00489
               template <typename T>
00490
               using push_front = type_list<T, Ts...>;
00491
00494
               template <size_t index>
00495
               using at = internal::type_at_t<index, Ts...>;
00496
00498
               struct pop_front {
00500
                   using type = typename internal::pop_front_h<Ts...>::head;
00502
                   using tail = typename internal::pop_front_h<Ts...>::tail;
00503
00504
00507
               template <typename T>
00508
               using push_back = type_list<Ts..., T>;
00509
00512
               template <typename U>
00513
               using concat = typename concat_h<U>::type;
00514
00517
               template <size_t index>
00518
               struct split {
00519
               private:
00520
                   using inner = internal::split_h<index, type_list<>, type_list<Ts...»;</pre>
00521
00522
                public:
00523
                   using head = typename inner::head;
00524
                   using tail = typename inner::tail;
00525
00526
00530
               template <typename T, size_t index>
00531
               using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00532
00535
               template <size_t index>
00536
               using remove = typename internal::remove_h<index, type_list<Ts...»::type;
00537
          };
00538
00540
          template <>
00541
          struct type_list<> {
00542
               static constexpr size_t length = 0;
00543
00544
               template <typename T>
00545
               using push_front = type_list<T>;
00546
00547
               template <typename T>
00548
               using push_back = type_list<T>;
00549
00550
               template <typename U>
00551
               using concat = U;
00552
               // TODO(jewave): assert index == 0
00553
00554
               template <typename T, size_t index>
using insert = type_list<T>;
00555
00557 } // namespace aerobus
00558
00559 // i32
00560 namespace aerobus {
00562
          struct i32 {
```

```
using inner_type = int32_t;
00566
              template<int32_t x>
00567
              struct val {
                  using enclosing_type = i32;
static constexpr int32_t v = x;
00569
00571
00572
00575
                  template<typename valueType>
00576
                  static constexpr INLINED DEVICE valueType get() { return static_cast<valueType>(x); }
00577
00579
                  using is_zero_t = std::bool_constant<x == 0>;
00580
00582
                  static std::string to string() {
00583
                      return std::to string(x);
00584
00585
00588
                  template<typename valueRing>
                  static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
00589
00590
                      return static_cast<valueRing>(x);
00591
00592
              };
00593
00595
              using zero = val<0>;
00597
              using one = val<1>;
00599
              static constexpr bool is_field = false;
00601
              static constexpr bool is_euclidean_domain = true;
00605
              template<auto x>
00606
              using inject_constant_t = val<static_cast<int32_t>(x)>;
00607
00608
              template < typename v >
00609
              using inject_ring_t = v;
00610
00611
           private:
00612
              template<typename v1, typename v2>
00613
              struct add {
00614
                  using type = val<v1::v + v2::v>;
00615
00616
00617
              template<typename v1, typename v2>
00618
              struct sub {
00619
                 using type = val<v1::v - v2::v>;
00620
00621
00622
              template<typename v1, typename v2>
00623
              struct mul {
00624
                  using type = val<v1::v* v2::v>;
00625
00626
00627
              template<typename v1, typename v2>
00628
              struct div {
00629
                 using type = val<v1::v / v2::v>;
00630
              };
00631
00632
              template<typename v1, typename v2>
00633
              struct remainder {
                  using type = val<v1::v % v2::v>;
00634
00635
              };
00636
00637
              template<typename v1, typename v2>
00638
00639
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00640
00641
00642
              template<typename v1, typename v2>
00643
              struct lt {
00644
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00645
00646
00647
              template<typename v1, typename v2>
00648
              struct ea {
                 using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00649
00650
00651
00652
              template<typename v1>
00653
              struct pos {
                  using type = std::bool_constant<(v1::v > 0)>;
00654
00655
00656
00657
           public:
00663
              template<typename v1, typename v2>
00664
              using add_t = typename add<v1, v2>::type;
00665
00671
              template<typename v1, typename v2>
00672
              using sub_t = typename sub<v1, v2>::type;
00673
00679
              template<typename v1, typename v2> \,
00680
              using mul_t = typename mul<v1, v2>::type;
00681
```

```
template<typename v1, typename v2>
00688
              using div_t = typename div<v1, v2>::type;
00689
00695
              template<typename v1, typename v2>
00696
              using mod t = typename remainder < v1, v2>::type;
00697
00703
              template<typename v1, typename v2>
00704
              using gt_t = typename gt<v1, v2>::type;
00705
00711
              template<typename v1, typename v2>
00712
              using lt_t = typename lt<v1, v2>::type;
00713
              template<typename v1, typename v2>
00719
00720
              using eq_t = typename eq<v1, v2>::type;
00721
00726
              template<typename v1, typename v2>
              static constexpr bool eq_v = eq_t<v1, v2>::value;
00727
00728
              template<typename v1, typename v2>
00735
             using gcd_t = gcd_t < i32, v1, v2>;
00736
00741
              template<typename v>
00742
              using pos_t = typename pos<v>::type;
00743
00748
              template<typename v>
00749
             static constexpr bool pos_v = pos_t<v>::value;
00750
00751 } // namespace aerobus
00752
00753 // i64
00754 namespace aerobus {
         struct i64 {
00758
           using inner_type = int64_t;
00761
              template<int64_t x>
00762
              struct val {
00764
                 using inner_type = int32_t;
00766
                  using enclosing_type = i64;
00768
                 static constexpr int64_t v = x;
00769
00772
                 template<typename valueType>
00773
                  static constexpr DEVICE INLINED valueType get() {
00774
                      return static_cast<valueType>(x);
00775
00776
00778
                  using is_zero_t = std::bool_constant<x == 0>;
00779
00781
                  static std::string to_string() {
00782
                      return std::to_string(x);
00783
00784
00787
                  template<typename valueRing>
00788
                  static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
00789
                      return static_cast<valueRing>(x);
00790
00791
              };
00792
00796
              template<auto x>
              using inject_constant_t = val<static_cast<int64_t>(x)>;
00797
00798
00803
              template<typename v>
00804
              using inject_ring_t = v;
00805
00807
              using zero = val<0>;
00809
              using one = val<1>;
00811
              static constexpr bool is_field = false;
00813
              static constexpr bool is_euclidean_domain = true;
00814
00815
           private:
              template<typename v1, typename v2>
00816
00817
              struct add {
00818
                 using type = val<v1::v + v2::v>;
00819
00820
00821
              template<typename v1, typename v2>
00822
              struct sub {
00823
                  using type = val<v1::v - v2::v>;
00824
00825
00826
              template<typename v1, typename v2>
00827
              struct mul {
                 using type = val<v1::v* v2::v>;
00828
00829
00830
00831
              template<typename v1, typename v2>
              struct div {
00832
                  using type = val<v1::v / v2::v>;
00833
00834
              };
```

```
00835
00836
               template<typename v1, typename v2>
00837
               struct remainder {
00838
                 using type = val<v1::v% v2::v>;
00839
00840
00841
               template<typename v1, typename v2>
00842
00843
                 using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00844
00845
00846
               template<typename v1, typename v2>
00847
               struct lt {
00848
                   using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00849
00850
00851
               template<typename v1, typename v2>
00852
               struct eq {
00853
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00854
               };
00855
00856
               template<typename v>
00857
               struct pos {
                   using type = std::bool_constant<(v::v > 0)>;
00858
00859
               };
00860
00861
           public:
00866
               template<typename v1, typename v2>
00867
               using add_t = typename add<v1, v2>::type;
00868
               template<typename v1, typename v2>
00873
00874
               using sub_t = typename sub<v1, v2>::type;
00875
00880
               template<typename v1, typename v2>
00881
               using mul_t = typename mul<v1, v2>::type;
00882
00888
              template<typename v1, typename v2>
using div_t = typename div<v1, v2>::type;
00890
00895
               template<typename v1, typename v2>
00896
               using mod_t = typename remainder<v1, v2>::type;
00897
00903
               template<typename v1, typename v2>
00904
               using qt_t = typename qt<v1, v2>::type;
00905
               template<typename v1, typename v2>
00910
00911
               static constexpr bool gt_v = gt_t<v1, v2>::value;
00912
00918
              template<typename v1, typename v2>
using lt_t = typename lt<v1, v2>::type;
00919
00920
               template<typename v1, typename v2>
static constexpr bool lt_v = lt_t<v1, v2>::value;
00926
00927
00928
00934
               template<typename v1, typename v2>
00935
              using eq_t = typename eq<v1, v2>::type;
00936
00942
               template<typename v1, typename v2>
00943
               static constexpr bool eq_v = eq_t<v1, v2>::value;
00944
00950
               template<typename v1, typename v2>
00951
              using gcd_t = gcd_t < i64, v1, v2>;
00952
00957
               template<typename v>
00958
               using pos_t = typename pos<v>::type;
00959
00964
               template<typename v>
00965
               static constexpr bool pos_v = pos_t<v>::value;
00966
          };
00967 } // namespace aerobus
00968
00969 // z/pz
00970 namespace aerobus {
00975
          template<int32_t p>
00976
          struct zpz {
00977
              using inner_type = int32_t;
               template<int32_t x>
00978
00979
               struct val {
                  using enclosing_type = zpz;
static constexpr int32_t v = x % p;
00981
00983
00984
00985
                   template<typename valueType>
00986
                   static constexpr DEVICE INLINED valueType get() { return static_cast<valueType>(x % p); }
00987
00988
                   using is_zero_t = std::bool_constant<x% p == 0>;
00989
                   static std::string to_string() {
00990
                       return std::to_string(x % p);
```

```
}
00992
00993
                  template<typename valueRing>
                  static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
00994
00995
                      return static_cast<valueRing>(x % p);
00996
                  }
00997
              } ;
00998
00999
              template<auto x>
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01000
01001
01002
              using zero = val<0>:
01003
              using one = val<1>;
01004
              static constexpr bool is_field = is_prime::value;
01005
              static constexpr bool is_euclidean_domain = true;
01006
           private:
01007
01008
              template<typename v1, typename v2>
01009
              struct add {
01010
                 using type = val<(v1::v + v2::v) % p>;
01011
01012
01013
              template<typename v1, typename v2>
01014
              struct sub {
01015
                  using type = val<(v1::v - v2::v) % p>;
01016
01017
01018
              template<typename v1, typename v2>
01019
              struct mul {
                  using type = val<(v1::v* v2::v) % p>;
01020
01021
01022
01023
              template<typename v1, typename v2>
              struct div {
01024
01025
                  using type = val<(v1::v% p) / (v2::v % p)>;
01026
01027
              template<typename v1, typename v2>
01029
              struct remainder {
01030
                 using type = val<(v1::v% v2::v) % p>;
01031
01032
01033
              template<typename v1, typename v2>
01034
              struct qt {
01035
                 using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01036
01037
01038
              template<typename v1, typename v2>
01039
              struct lt {
                 using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01040
01041
01042
01043
              template<typename v1, typename v2>
01044
              struct eq {
01045
                  using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01046
              };
01047
01048
              template<typename v1>
01049
              struct pos {
01050
                  using type = std::bool_constant<(v1::v > 0)>;
              }:
01051
01052
01053
           public:
01057
              template<typename v1, typename v2>
01058
              using add_t = typename add<v1, v2>::type;
01059
01063
              template<typename v1, typename v2>
01064
              using sub_t = typename sub<v1, v2>::type;
01065
              template<typename v1, typename v2>
01070
              using mul_t = typename mul<v1, v2>::type;
01071
01075
              template<typename v1, typename v2>
01076
              using div_t = typename div<v1, v2>::type;
01077
01081
              template<typename v1, typename v2>
01082
              using mod_t = typename remainder<v1, v2>::type;
01083
01087
              template<typename v1, typename v2>
01088
              using gt_t = typename gt<v1, v2>::type;
01089
01093
              template<typename v1, typename v2>
01094
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01095
01099
              template<typename v1, typename v2> ^{\circ}
01100
              using lt_t = typename lt<v1, v2>::type;
01101
```

```
template<typename v1, typename v2>
              static constexpr bool lt_v = lt_t<v1, v2>::value;
01106
01107
01111
              template<typename v1, typename v2>
01112
              using eq_t = typename eq<v1, v2>::type;
01113
01117
              template<typename v1, typename v2>
01118
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01119
01123
              template<typename v1, typename v2>
              using gcd_t = gcd_t < i32, v1, v2>;
01124
01125
01128
              template<typename v1>
01129
              using pos_t = typename pos<v1>::type;
01130
01133
              template<typename v>
01134
              static constexpr bool pos_v = pos_t < v > :: value;
01135
01136 } // namespace aerobus
01138 // polynomial
01139 namespace aerobus {
         // coeffN x^N + ...
01140
01145
         template<typename Ring>
01146
          requires IsEuclideanDomain<Ring>
01147
         struct polynomial {
01148
              static constexpr bool is_field = false;
01149
              static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01150
01154
              template<typename coeffN, typename... coeffs>
01155
              struct val {
01157
                  using enclosing_type = polynomial<Ring>;
01159
                  static constexpr size_t degree = sizeof...(coeffs);
01161
                  using aN = coeffN;
01163
                  using strip = val<coeffs...>;
                  using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01165
                  static constexpr bool is_zero_v = is_zero_t::value;
01167
01168
01169
01170
                  template<size_t index, typename E = void>
01171
                  struct coeff_at {};
01172
01173
                  template<size t index>
01174
                  struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))» {</pre>
01175
                      using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01176
01177
01178
                  template<size_t index>
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))» {
01179
01180
                     using type = typename Ring::zero;
01181
                  };
01182
01183
               public:
01186
                  template<size_t index>
                  using coeff_at_t = typename coeff_at<index>::type;
01187
01188
                  static std::string to_string() {
01192
                      return string_helper<coeffN, coeffs...>::func();
01193
01194
01199
                  template<typename valueRing>
                  static constexpr DEVICE INLINED valueRing eval(const valueRing& x) {
01200
01201
                      return horner_evaluation<valueRing, val>
01202
                              ::template inner<0, degree + 1>
01203
                              ::func(static_cast<valueRing>(0), x);
01204
                  }
01205
              };
01206
01209
              template<tvpename coeffN>
              struct val<coeffN> {
01212
                  using enclosing_type = polynomial<Ring>;
01214
                  static constexpr size_t degree = 0;
01215
                  using aN = coeffN;
                  using strip = val<coeffN>:
01216
                  using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01217
01218
01219
                  static constexpr bool is_zero_v = is_zero_t::value;
01220
01221
                  template<size_t index, typename E = void>
01222
                  struct coeff at {};
01223
01224
                  template<size_t index>
01225
                  struct coeff_at<index, std::enable_if_t<(index == 0)» {</pre>
01226
                      using type = aN;
01227
01228
01229
                  template<size t index>
```

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

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

```
>::value && (P::degree > 0)
01408
01409
                  using type = P;
01410
              };
01411
              // do not simplify constants
01412
01413
              template<typename P>
01414
              struct simplify<P, std::enable_if_t<P::degree == 0» {</pre>
01415
                 using type = P;
01416
01417
              // addition at
01418
01419
              template<typename P1, typename P2, size_t index>
01420
              struct add_at {
01421
                  using type =
01422
                      typename Ring::template add_t<</pre>
                          typename P1::template coeff_at_t<index>,
01423
01424
                          typename P2::template coeff_at_t<index>>;
01426
01427
              template<typename P1, typename P2, size_t index>
01428
              using add_at_t = typename add_at<P1, P2, index>::type;
01429
              template<typename P1, typename P2, std::size_t... I>
struct add_low<P1, P2, std::index_sequence<I...» {</pre>
01430
01431
                 using type = val<add_at_t<P1, P2, I>...>;
01432
01433
01434
              // substraction at
01435
              template<typename P1, typename P2, size_t index>
01436
01437
              struct sub at {
01438
                  using type :
01439
                      typename Ring::template sub_t<</pre>
01440
                          typename P1::template coeff_at_t<index>,
01441
                          typename P2::template coeff_at_t<index>>;
01442
01443
              template<typename P1, typename P2, size_t index>
              using sub_at_t = typename sub_at<P1, P2, index>::type;
01445
01446
01447
              template<typename P1, typename P2, std::size_t... I>
              struct sub_low<P1, P2, std::index_sequence<I...» {
01448
01449
                 using type = val<sub at t<P1, P2, I>...>;
01450
01451
01452
              template<typename P1, typename P2>
01453
              struct sub {
01454
                  using type = typename simplify<typename sub_low<
01455
                  Р1.
01456
01457
                  internal::make_index_sequence_reverse<</pre>
01458
                  std::max(P1::degree, P2::degree) + 1
01459
                  »::type>::type;
01460
              };
01461
              // multiplication at
01462
              template<typename v1, typename v2, size_t k, size_t index, size_t stop>
01464
              struct mul_at_loop_helper {
01465
                  using type = typename Ring::template add_t<
                      typename Ring::template mul_t<
typename v1::template coeff_at_t<index>,
01466
01467
01468
                      typename v2::template coeff_at_t<k - index>
01469
01470
                       typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
01471
01472
              };
01473
01474
              template<typename v1, typename v2, size_t k, size_t stop>
01475
              struct mul_at_loop_helper<v1, v2, k, stop, stop> {
01476
                  using type = typename Ring::template mul_t<
01477
                     typename v1::template coeff_at_t<stop>,
01478
                       typename v2::template coeff_at_t<0>>;
01479
              };
01480
              template <typename v1, typename v2, size_t k, typename E = void>
01481
              struct mul_at {};
01482
01483
01484
              template<typename v1, typename v2, size_t k>
              01485
01486
                  using type = typename Ring::zero;
01487
01488
              template<typename v1, typename v2, size_t k> struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)» {
01489
01490
01491
                  using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01492
01493
```

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

```
std::string tail = string_helper<coeffs...>::func();
                       std::string result = "";
01577
01578
                       if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
                       return tail;
} else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
01579
01580
                           if (sizeof...(coeffs) == 1) {
    result += "x";
01581
01582
01583
                           } else {
01584
                              result += "x^" + std::to_string(sizeof...(coeffs));
01585
                           }
01586
                       } else {
                           if (sizeof...(coeffs) == 1) {
01587
01588
                               result += coeff::to_string() + " x";
01589
                           } else {
01590
                               result += coeff::to_string()
01591
                                       + " x^" + std::to_string(sizeof...(coeffs));
01592
                           }
01593
                       }
01594
                       if (!tail.empty()) {
    result += " + " + tail;
01595
01596
01597
01598
01599
                       return result:
01600
                  }
01601
              };
01602
01603
              template<typename coeff>
01604
              struct string_helper<coeff> {
01605
                  static std::string func() {
01606
                      if (!std::is_same<coeff, typename Ring::zero>::value) {
01607
                           return coeff::to_string();
01608
                      } else {
01609
                           return "";
                       }
01610
01611
                  }
              };
01612
01613
01614
01617
              template<typename P>
01618
              using simplify_t = typename simplify<P>::type;
01619
01623
              template<typename v1, typename v2>
01624
              using add_t = typename add<v1, v2>::type;
01625
01629
              template<typename v1, typename v2>
01630
              using sub_t = typename sub<v1, v2>::type;
01631
01635
              template<typename v1, typename v2>
01636
              using mul t = typename mul<v1, v2>::type;
01637
01641
              template<typename v1, typename v2>
01642
              using eq_t = typename eq_helper<v1, v2>::type;
01643
              template<typename v1, typename v2>
01647
01648
              using lt_t = typename lt_helper<v1, v2>::type;
01649
01653
              template<typename v1, typename v2>
01654
              using gt_t = typename gt_helper<v1, v2>::type;
01655
              template<typename v1, typename v2>
01659
01660
              using div t = typename div<v1, v2>::g type;
01661
01665
              template<typename v1, typename v2>
01666
              using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
01667
01671
              template<typename coeff, size_t deg>
              using monomial_t = typename monomial<coeff, deg>::type;
01672
01673
              template<typename v>
01677
              using derive_t = typename derive_helper<v>::type;
01678
01681
              template<typename v>
01682
              using pos_t = typename Ring::template pos_t<typename v::aN>;
01683
01686
              template<typename v>
01687
              static constexpr bool pos_v = pos_t<v>::value;
01688
01692
              template<typename v1, typename v2>
01693
              using gcd t = std::conditional t<
01694
                 Ring::is euclidean domain,
01695
                  typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
01696
                   void>;
01697
01701
              template<auto x>
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
01702
01703
```

```
template<typename v>
01708
                          using inject_ring_t = val<v>;
01709
01710 } // namespace aerobus
01711
01712 // fraction field
01713 namespace aerobus {
01714
                  namespace internal {
01715
                        template<typename Ring, typename E = void>
01716
                          requires IsEuclideanDomain<Ring>
                          struct _FractionField {};
01717
01718
01719
                          template<typename Ring>
01720
                          requires IsEuclideanDomain<Ring>
01721
                          struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain> {
01723
                                  static constexpr bool is_field = true;
                                  static constexpr bool is_euclidean_domain = true;
01724
01725
01726
01727
                                 template<typename val1, typename val2, typename E = void>
01728
                                 struct to_string_helper {};
01729
01730
                                 template<typename val1, typename val2>
                                 struct to_string_helper <val1, val2,
    std::enable_if_t<</pre>
01731
01732
01733
                                         Ring::template eq_t<
01734
                                         val2, typename Ring::one
01735
                                         >::value
01736
                                         >
01737
                                 > {
01738
                                         static std::string func() {
                                                return vall::to string();
01740
01741
                                  } ;
01742
                                  template<typename val1, typename val2>
01743
01744
                                  struct to_string_helper<val1, val2,
01745
                                         std::enable_if_t<
01746
                                          !Ring::template eq_t<
01747
                                         val2,
01748
                                         typename Ring::one
01749
                                         >::value
01750
                                         >
01751
                                 > {
01752
                                         static std::string func() {
01753
                                                return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
01754
01755
                                 };
01756
01757
                            public:
01761
                                 template<typename val1, typename val2>
01762
                                  struct val {
01764
                                         using x = val1;
                                         using y = val2;
01766
01768
                                         using is_zero_t = typename vall::is_zero_t;
01770
                                         static constexpr bool is zero v = val1::is zero t::value;
01771
01773
                                         using ring_type = Ring;
01774
                                         using enclosing_type = _FractionField<Ring>;
01775
01778
                                           static constexpr bool is integer = std::is same v<val2, typename Ring::one>;
01779
01783
                                         template<typename valueType>
01784
                                         static constexpr DEVICE INLINED valueType get() {
01785
                                                 return static_cast<valueType>(x::v) / static_cast<valueType>(y::v);
01786
01787
01790
                                         static std::string to_string() {
01791
                                                return to_string_helper<val1, val2>::func();
01792
01793
01798
                                         template<typename valueRing>
                                         static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
    return x::eval(v) / y::eval(v);
01799
01800
01801
01802
                                 };
01803
01805
                                  using zero = val<typename Ring::zero, typename Ring::one>;
01807
                                  using one = val<typename Ring::one, typename Ring::one>;
01808
01811
                                 template<typename v>
01812
                                 using inject_t = val<v, typename Ring::one>;
01813
01816
01817
                                 \verb"using inject_constant_t = \verb"val<typename Ring::template inject_constant_t < \verb"x">x>, typename Ring::template inject_constant_t < x>, typename Ring::template inject_consta
           Ring::one>;
01818
```

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

```
template<typename v1, typename v2, typename E = void>
01910
01911
01912
                  template<typename v1, typename v2> \,
01913
                  struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
      _FractionField<Ring>::zero>::value»
01914
                   private:
01915
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01916
                      using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01917
01918
                   public:
                      using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
01919
01920
01921
01922
                  template<typename v1, typename v2>
01923
                  struct div<v1, v2, std::enable_if_t<
01924
                      std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
01925
                      using type = one;
01926
01927
01928
                  template<typename v1, typename v2>
01929
                  struct eq {
01930
                      using type = std::conditional_t<
                              std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
01931
01932
                              std::is_same<typename simplify_t<vl>::y, typename simplify_t<v2>::y>::value,
01933
                          std::true_type,
01934
                          std::false_type>;
01935
                  };
01936
01937
                  template<typename v1, typename v2, typename E = void>
01938
                  struct at:
01939
01940
                  template<typename v1, typename v2>
01941
                  struct gt<v1, v2, std::enable_if_t<
01942
                      (eq<v1, v2>::type::value)
01943
01944
                      using type = std::false_type;
01945
                  };
01946
01947
                  template<typename v1, typename v2>
01948
                  struct gt<v1, v2, std::enable_if_t<
                      (!eq<v1, v2>::type::value) &&
01949
                      (!pos<v1>::type::value) && (!pos<v2>::type::value)
01950
01951
01952
                      using type = typename gt<
01953
                          typename sub<zero, v1>::type, typename sub<zero, v2>::type
01954
01955
                  };
01956
01957
                  template<tvpename v1, tvpename v2>
                  struct gt<v1, v2, std::enable_if_t<
01958
01959
                       (!eq<v1, v2>::type::value) &&
01960
                       (pos<v1>::type::value) && (!pos<v2>::type::value)
01961
                      using type = std::true_type;
01962
01963
                  };
01964
01965
                  template<typename v1, typename v2>
01966
                  struct gt<v1, v2, std::enable_if_t<
                      (!eq<v1, v2>::type::value) &&
01967
01968
                      01969
                      » {
01970
                      using type = std::false_type;
01971
01972
01973
                  template<typename v1, typename v2>
01974
                  struct gt<v1, v2, std::enable_if_t<
    (!eq<v1, v2>::type::value) &&
01975
01976
                      (pos<v1>::tvpe::value) && (pos<v2>::tvpe::value)
01977
01978
                      using type = typename Ring::template gt_t<
01979
                           typename Ring::template mul_t<v1::x, v2::y>,
01980
                          typename Ring::template mul_t<v2::y, v2::x>
01981
01982
                  };
01983
01984
               public:
01989
                  template<typename v1, typename v2>
01990
                  using add_t = typename add<v1, v2>::type;
01991
                  template<typename v1, typename v2>
01996
01997
                  using mod_t = zero;
01998
02003
                  template<typename v1, typename v2>
02004
                  using gcd_t = v1;
02005
02009
                  template<tvpename v1, tvpename v2>
```

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

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

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

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

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

```
struct cos_coeff {
02547
                  using type = typename cos_coeff_helper<T, i>::type;
02548
              };
02549
              template<typename T, size_t i, typename E = void>
02550
02551
              struct cosh coeff helper {};
02552
02553
               template<typename T, size_t i>
02554
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02555
                   using type = typename FractionField<T>::zero;
02556
              };
02557
02558
              template<typename T, size_t i>
02559
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02560
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02561
02562
02563
              template<typename T, size_t i>
              struct cosh_coeff {
02564
02565
                  using type = typename cosh_coeff_helper<T, i>::type;
02566
02567
02568
              template<typename T, size_t i>
              struct geom_coeff { using type = typename FractionField<T>::one; };
02569
02570
02571
02572
              template<typename T, size_t i, typename E = void>
02573
              struct atan_coeff_helper;
02574
02575
              template<typename T, size_t i>
02576
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
                  using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;;
02578
02579
              template<typename T, size_t i>
02580
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
02581
02582
02584
02585
              template<typename T, size_t i>
02586
              struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
02587
              template<typename T, size_t i, typename E = void>
02588
02589
              struct asin_coeff_helper;
02590
02591
               template<typename T, size_t i>
02592
               struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
02593
                   using type = makefraction_t<T,</pre>
                       factorial_t<T, i - 1>,
02594
                       typename T::template mul_t<
02595
02596
                            typename T::template val<i>,
02597
                           T::template mul_t<
02598
                               pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
02599
                               pow<T, factorial_t<T, i / 2>, 2
02600
02601
02602
                       »;
02603
              };
02604
02605
              template<typename T, size_t i>
              struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
02606
02607
02608
02609
02610
              template<typename T, size_t i>
02611
              struct asin_coeff {
02612
                  using type = typename asin_coeff_helper<T, i>::type;
02613
02614
              template<typename T, size_t i>
02616
              struct lnp1_coeff {
02617
                   using type = makefraction_t<T,</pre>
                       alternate_t<T, i + 1>,
02618
02619
                       typename T::template val<i>;;
02620
              };
02621
02622
               template<typename T>
02623
               struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
02624
02625
              template<typename T, size t i, typename E = void>
02626
              struct asinh_coeff_helper;
02627
02628
               template<typename T, size_t i>
02629
               struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02630
                  using type = makefraction_t<T,</pre>
02631
                       typename T::template mul_t<
02632
                           alternate t<T, i / 2>,
```

```
factorial_t<T, i - 1>
02634
02635
                                          typename T::template mul_t<
02636
                                                  typename T::template mul_t<</pre>
                                                         typename T::template val<i>,
pow_t<T, factorial_t<T, i / 2>, 2>
02637
02638
02639
02640
                                                  pow_t<T, typename T::template inject_constant_t<4>, i / 2>
02641
02642
                                  >;
                          };
02643
02644
02645
                           template<typename T, size_t i>
02646
                           struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02647
                                 using type = typename FractionField<T>::zero;
02648
02649
02650
                           template<typename T, size_t i>
                           struct asinh_coeff {
02651
02652
                                 using type = typename asinh_coeff_helper<T, i>::type;
02653
02654
02655
                           template<typename T, size_t i, typename E = void>
02656
                           struct atanh coeff helper;
02657
02658
                           template<typename T, size_t i>
02659
                           struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02660
                                  // 1/i
02661
                                  using type = typename FractionField<T>:: template val<</pre>
02662
                                          typename T::one,
02663
                                          typename T::template inject_constant_t<i>;;
02664
                           };
02665
02666
                           template<typename T, size_t i>
02667
                           struct \ atanh\_coeff\_helper<T, \ i, \ std::enable\_if\_t<(i \& 1) == 0 > \{truct \ atanh\_coeff\_helper<T, \ i, \ std::enable\_if\_t<(i \& 1) == 0 > (i \& 1) == 0 >
                                  using type = typename FractionField<T>::zero;
02668
02669
                           };
02670
02671
                           template<typename T, size_t i>
02672
                           struct atanh_coeff {
02673
                                  using type = typename atanh_coeff_helper<T, i>::type;
02674
02675
02676
                           template<typename T, size_t i, typename E = void>
02677
                          struct tan_coeff_helper;
02678
02679
                           template<typename T, size_t i>
                           struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
    using type = typename FractionField<T>::zero;
02680
02681
02682
02683
02684
                           template<typename T, size_t i>
02685
                           struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
02686
                           private:
                                  // 4^((i+1)/2)
02687
                                  using _4p = typename FractionField<T>::template inject_t<</pre>
02688
02689
                                         pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
02690
                                   // 4^((i+1)/2)
                                  using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename</pre>
02691
          FractionField<T>::one>;
02692
                                  //(-1)^{(i-1)/2}
                                  using dividend = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»; using dividend = typename FractionField<T>::template mul_t<
02693
02694
02695
                                          altp,
02696
                                          FractionField<T>::template mul_t<
02697
                                          FractionField<T>::template mul_t<
02698
02699
                                           4pm1.
02700
                                          bernoulli_t<T, (i + 1)>
02701
02702
02703
                           public:
02704
                                 using type = typename FractionField<T>::template div_t<dividend,</pre>
02705
02706
                                          typename FractionField<T>::template inject t<factorial t<T, i + 1>>;
02707
                          };
02708
02709
                           template<typename T, size_t i>
02710
                           struct tan_coeff {
02711
                                 using type = typename tan_coeff_helper<T, i>::type;
02712
02714
                           template<typename T, size_t i, typename E = void>
02715
                           struct tanh_coeff_helper;
02716
                           template<typename T, size_t i>
struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {</pre>
02717
02718
```

```
using type = typename FractionField<T>::zero;
02720
02721
02722
              template<typename T, size_t i>
              struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
02723
02724
              private:
02725
                 using _4p = typename FractionField<T>::template inject_t<</pre>
02726
                      pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
02727
                  using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename</pre>
     FractionField<T>::one>;
02728
                  using dividend =
02729
                      typename FractionField<T>::template mul_t<</pre>
02730
                          _4p,
typename FractionField<T>::template mul_t<
02731
02732
                              _4pm1,
02733
                              bernoulli_t<T, (i + 1) >>::type;
              public:
02734
02735
                 using type = typename FractionField<T>::template div_t<dividend,</pre>
                      FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02737
              };
02738
02739
              template<typename T, size_t i>
02740
              struct tanh_coeff {
                  using type = typename tanh_coeff_helper<T, i>::type;
02741
02742
02743
         } // namespace internal
02744
02748
          template<typename Integers, size_t deg>
02749
          using exp = taylor<Integers, internal::exp_coeff, deg>;
02750
02754
          template<typename Integers, size_t deg>
          using expm1 = typename polynomial<FractionField<Integers>>::template sub_t<</pre>
02755
02756
              exp<Integers, deg>,
02757
              typename polynomial<FractionField<Integers>>::one>;
02758
02762
          template<typename Integers, size_t deg>
02763
          using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
02764
02768
          template<typename Integers, size_t deg>
02769
          using atan = taylor<Integers, internal::atan_coeff, deg>;
02770
02774
          template<typename Integers, size_t deg>
02775
          using sin = taylor<Integers, internal::sin coeff, deg>;
02776
02780
          template<typename Integers, size_t deg>
02781
          using sinh = taylor<Integers, internal::sh_coeff, deg>;
02782
02787
          template<typename Integers, size_t deg>
02788
          using cosh = taylor<Integers, internal::cosh_coeff, deg>;
02789
02794
          template<typename Integers, size_t deg>
02795
          using cos = taylor<Integers, internal::cos_coeff, deg>;
02796
02801
          template<typename Integers, size_t deg>
          using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
02802
02803
02808
          template<typename Integers, size_t deg>
02809
          using asin = taylor<Integers, internal::asin_coeff, deg>;
02810
02815
          template<typename Integers, size_t deg>
          using asinh = taylor<Integers, internal::asinh_coeff, deg>;
02816
02817
02822
          template<typename Integers, size_t deg>
          using atanh = taylor<Integers, internal::atanh_coeff, deg>;
02823
02824
02829
          template<typename Integers, size_t deg>
02830
          using tan = taylor<Integers, internal::tan_coeff, deg>;
02831
          template<typename Integers, size_t deg>
02836
          using tanh = taylor<Integers, internal::tanh_coeff, deg>;
02838 }
         // namespace aerobus
02839
02840 // continued fractions
02841 namespace aerobus {
         template<int64_t... values>
02850
02851
          struct ContinuedFraction {};
02852
02855
          template<int64_t a0>
          struct ContinuedFraction<a0> {
02856
              using type = typename q64::template inject_constant_t<a0>;
02858
02860
              static constexpr double val = static_cast<double>(a0);
02861
          };
02862
02866
          template<int64_t a0, int64_t... rest>
02867
          struct ContinuedFraction<a0, rest...> {
02869
              using type = q64::template add_t<
02870
                      typename q64::template inject_constant_t<a0>,
```

```
typename q64::template div_t<
02872
                                            typename q64::one,
02873
                                            typename ContinuedFraction<rest...>::type
02874
                       static constexpr double val = type::template get<double>();
02876
02877
                 };
02878
02883
                 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>;
02886
                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>;
02888
               using SORT2 fraction =
          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, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 
          // NOLINT
02891 }
              // namespace aerobus
02892
02893 // known polynomials
02894 namespace aerobus {
02895
                 // CChebyshev
02896
                 namespace internal {
02897
                       template<int kind, size_t deg, typename I>
                        struct chebyshev_helper {
02898
02899
                              using type = typename polynomial<I>::template sub_t<
02900
                                     typename polynomial<I>::template mul_t<</pre>
02901
                                            typename polynomial<I>::template mul_t<</pre>
02902
                                                   typename polynomial<I>::template inject_constant_t<2>,
02903
                                                   typename polynomial<I>::X>
02904
                                            typename chebyshev_helper<kind, deg - 1, I>::type
02905
02906
                                     typename chebyshev_helper<kind, deg - 2, I>::type
02907
02908
                       } ;
02909
02910
                       template<typename I>
02911
                       struct chebyshev_helper<1, 0, I> {
02912
                              using type = typename polynomial<I>::one;
02913
02914
02915
                       template<typename I>
                       struct_chebyshev_helper<1, 1, I> {
02916
02917
                             using type = typename polynomial<I>::X;
02918
02919
02920
                        template<typename I>
02921
                        struct chebyshev_helper<2, 0, I> {
02922
                              using type = typename polynomial<I>::one;
02923
02924
02925
                       template<typename I>
02926
                        struct chebyshev_helper<2, 1, I> {
02927
                              using type = typename polynomial<I>::template mul_t<</pre>
02928
                                     typename polynomial<I>::template inject_constant_t<2>,
02929
                                     typename polynomial<I>::X>;
02930
02931
                 } // namespace internal
02932
02933
                 // Laguerre
02934
                 namespace internal {
02935
                       template<size_t deg, typename I>
02936
                       struct laguerre_helper {
                              using Q = FractionField<I>;
using PQ = polynomial<Q>;
02937
02938
02939
                         private:
02940
02941
                              // Lk = (1 / k) * ((2 * k - 1 - x) * 1km1 - (k - 2)Lkm2)
                              using lnm2 = typename laguerre_helper<deg - 2, I>::type;
02942
                              using lnm1 = typename laguerre_helper<deg - 1, I>::type;
02943
                               // -x + 2k-1
02945
                               using p = typename PQ::template val<
02946
                                     typename Q::template inject_constant_t<-1>,
02947
                                     typename Q::template inject_constant_t<2 * deg - 1»;</pre>
                               // 1/n
02948
02949
                              using factor = typename PO::template inject ring t<
                                    typename Q::template val<typename I::one, typename I::template
         inject_constant_t<deg>>;
02951
02952
                         public:
                              using type = typename PQ::template mul t <
02953
02954
                                     factor,
02955
                                     typename PQ::template sub_t<
02956
                                            typename PQ::template mul_t<
02957
02958
                                                   1 nm1
02959
02960
                                            typename PQ::template mul_t<
```

```
02961
                              typename PQ::template inject_constant_t<deg-1>,
02962
02963
02964
                      >
02965
                  >;
02966
              };
02967
02968
              template<typename I>
02969
              struct laguerre_helper<0, I> {
02970
                  using type = typename polynomial<FractionField<I>::one;
02971
02972
02973
              template<typename I>
02974
              struct laguerre_helper<1, I> {
02975
              private:
02976
                  using PQ = polynomial<FractionField<I>;
02977
               public:
02978
                 using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
02979
              } ;
02980
          } // namespace internal
02981
02982
          // Bernstein
02983
          namespace internal {
02984
              template<size_t i, size_t m, typename I, typename E = void>
02985
              struct bernstein_helper {};
02986
02987
              template<typename I>
02988
              struct bernstein_helper<0, 0, I> {
02989
                 using type = typename polynomial<I>::one;
02990
02991
02992
              template<size_t i, size_t m, typename I>
              struct bernstein_helperi, m, I, std::enable_if_t<
(m > 0) && (i == 0) » {
02993
02994
               private:
02995
                 using P = polynomial<I>;
02996
               public:
02997
02998
                  using type = typename P::template mul_t<
02999
                          typename P::template sub_t<typename P::one, typename P::X>,
03000
                          typename bernstein_helper<i, m-1, I>::type>;
03001
03002
03003
              template<size_t i, size_t m, typename I>
              struct bernstein_helper<i, m, I, std::enable_if_t<
03004
                          (m > 0) && (i == m)  (
03005
03006
               private:
03007
                  using P = polynomial<I>;
03008
               public:
03009
                  using type = typename P::template mul_t<
03010
                          typename P::X,
03011
                          typename bernstein_helper<i-1, m-1, I>::type>;
03012
03013
03014
              template<size_t i, size_t m, typename I>
              03015
03016
03017
03018
                  using P = polynomial<I>;
03019
               public:
03020
                  using type = typename P::template add_t<
03021
                          typename P::template mul_t<
                              typename P::template sub_t<typename P::one, typename P::X>,
03022
03023
                              typename bernstein_helper<i, m-1, I>::type>,
03024
                          typename P::template mul_t<
03025
                              typename P::X,
03026
                              typename bernstein_helper<i-1, m-1, I>::type»;
03027
          };
} // namespace internal
03028
03029
03030
          namespace known_polynomials {
03032
             enum hermite_kind {
03034
                  probabilist,
03036
                  physicist
03037
              };
03038
          }
03039
03040
03041
          namespace internal {
03042
              template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03043
              struct hermite_helper {};
03044
03045
              template<size_t deg, typename I>
03046
              struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
              private:
03047
03048
                 using hnm1 = typename hermite_helper<deg - 1,
      known_polynomials::hermite_kind::probabilist, I>::type;
03049
                  using hnm2 = typename hermite_helper<deg - 2,
```

```
known_polynomials::hermite_kind::probabilist, I>::type;
03050
               public:
03051
03052
                   using type = typename polynomial<I>::template sub_t<
03053
                       \label{typename} typename polynomial < I>::template mul_t < typename polynomial < I>::X, hnml>, typename polynomial < I>::template mul_t <
03054
03055
                            typename polynomial<I>::template inject_constant_t<deg - 1>,
03056
03057
03058
03059
              };
03060
03061
               template<size_t deg, typename I>
03062
               struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03063
03064
                   using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
      I>::type;
03065
                   using hnm2 = typename hermite helper<deg - 2, known polynomials::hermite kind::physicist,
      I>::type;
03066
03067
               public:
03068
                   using type = typename polynomial<I>::template sub_t<</pre>
03069
                       // 2X Hn-1
03070
                       typename polynomial<I>::template mul_t<</pre>
                           typename pi64:val<typename I::template inject_constant_t<2>,
typename I::zero>, hnml>,
03071
03072
03073
                       typename polynomial<I>::template mul_t<</pre>
03074
03075
                            typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03076
                           hnm2
03077
03078
                   >;
03079
03080
03081
               template<typename I>
               struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03082
03083
                  using type = typename polynomial<I>::one;
03084
03085
03086
               template<typename I>
03087
               struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03088
                   using type = typename polynomial<I>::X;
03089
03090
03091
               template<typename I>
03092
               struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03093
                   using type = typename pi64::one;
03094
               };
03095
03096
               template<typename I>
03097
               struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03098
                   // 2X
03099
                   using type = typename polynomial<I>::template val<
03100
                       typename I::template inject_constant_t<2>,
                       typename I::zero>;
03101
03102
          } // namespace internal
03103
03104
03105
          // legendre
03106
          namespace internal {
03107
               template<size t n, typename I>
03108
               struct legendre_helper {
03109
               private:
03110
                  using Q = FractionField<I>;
03111
                   using PQ = polynomial<Q>;
                   // 1/n constant
// (2n-1)/n X
03112
03113
                   using fact_left = typename PQ::template monomial_t<
03114
03115
                       makefraction_t<I,
03116
                           typename I::template inject_constant_t<2*n-1>,
03117
                           typename I::template inject_constant_t<n>
                       >,
03118
                   1>;
// (n-1) / n
0.3119
03120
03121
                   using fact right = typename PO::template val<
03122
                       makefraction_t<I,
03123
                           typename I::template inject_constant_t<n-1>,
03124
                           typename I::template inject_constant_t<n>>;
03125
03126
                public:
                  using type = PQ::template sub_t<
03127
03128
                           typename PQ::template mul_t<
03129
03130
                                typename legendre_helper<n-1, I>::type
03131
                           typename PQ::template mul_t<
03132
03133
                                fact right.
```

```
typename legendre_helper<n-2, I>::type
03135
03136
                                    >;
03137
                       };
03138
03139
                       template<tvpename I>
03140
                       struct legendre_helper<0, I> {
03141
                              using type = typename polynomial<FractionField<I>::one;
03142
0.3143
03144
                       template<tvpename I>
                       struct legendre_helper<1, I> {
03145
                              using type = typename polynomial<FractionField<I»::X;
03146
03147
03148
                } // namespace internal
0.3149
                 // bernoulli polynomials
03150
03151
                namespace internal {
03152
                       template<size_t n>
03153
                       struct bernoulli_coeff {
03154
                              template<typename T, size_t i>
                              struct inner {
03155
                              private:
03156
                                    using F = FractionField<T>;
03157
                                public:
03158
03159
                                    using type = typename F::template mul_t<
03160
                                            typename F::template inject_ring_t<combination_t<T, i, n»,
03161
                                            bernoulli_t<T, n-i>
03162
03163
                              };
03164
                };
} // namespace internal
03165
03166
03168
                 namespace known_polynomials {
03176
                        template <size_t deg, typename I = aerobus::i64>
03177
                       using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
03178
03188
                       template <size_t deg, typename I = aerobus::i64>
03189
                       using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
03190
03200
                       template <size_t deg, typename I = aerobus::i64>
03201
                       using laguerre = typename internal::laguerre_helper<deg, I>::type;
03202
03209
                       template <size_t deg, typename I = aerobus::i64>
                       using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
03211
03218
                       template <size_t deg, typename I = aerobus::i64>
                       using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
03219
03220
                       template<size_t i, size_t m, typename I = aerobus::i64>
03232
                       using bernstein = typename internal::bernstein_helper<i, m, I>::type;
03233
03243
                       template<size_t deg, typename I = aerobus::i64>
03244
                       using legendre = typename internal::legendre_helper<deg, I>::type;
03245
03255
                       template<size_t deg, typename I = aerobus::i64>
03256
                       using bernoulli = taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
                     // namespace known_polynomials
03257
03258 } // namespace aerobus
03259
03260
03261 #ifdef AEROBUS_CONWAY_IMPORTS
03262
03263 // conway polynomials
03264 namespace aerobus {
03268
               template<int p, int n>
struct ConwayPolynomial {};
03269
03270
03271 #ifndef DO_NOT_DOCUMENT
03272
               #define ZPZV ZPZ::template val
03273
                 #define POLYV aerobus::polynomial<ZPZ>::template val
03274
                 template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
         POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
03275
                 template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
          POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
03276
                 template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
03277
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
         template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>; // NOLINT
03278
                 template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
03279
         POLYV<ZPZV<1>, 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<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
                template<> struct ConwayPolynomial<2, 7> { using ZPZ = aerobus::zpz<2>; using type =
03280
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>; // NOLINT template<> struct ConwayPolynomial<2, 8> { using ZPZ = aerobus::zpz<2>; using type =
03281
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                                                   template<> struct ConwayPolynomial<2, 9> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>; }; //
                                                   NOLINT
03283
                                                   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 , ZPZV<
                                                     ZPZV<1»; }; // NOLINT</pre>
                                                                                         template<> struct ConwayPolynomial<2, 11> { using ZPZ = aerobus::zpz<2>; using type
                                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
03285
                                                     ZPZV<0>, ZPZV<1>, ZPZV<1»; };</pre>
                                                                                                                                                                                                                                                                                                                                    // NOLINT
                                                                                        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<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , Z
                                                                                         template<> struct ConwayPolynomial<2, 15> { using ZPZ = aerobus::zpz<2>; using type
                                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                   ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT
    template<> struct ConwayPolynomial<2, 16> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03289
                                                     ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1</pre>
                                                                                         template<> struct ConwayPolynomial<2, 17> { 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<0>, ZPZV<0>, ZPZV<1>; }; // NOLINT
                                                   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<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZ
                                                                                         template<> struct ConwayPolynomial<2, 19> { 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<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>; }; //
                                                     NOLINT
                                                   template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , Z
03293
                                                                                        template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
                                                  POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
                                                                                      template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
03295
                                                   POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2»: }: // NOLINT
                                                                                      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 =
 03297
                                                POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; }; // NOLINT
                                                                                   template<> struct ConwayPolynomial<3, 5> { using ZPZ = aerobus::zpz<3>; using type =
 03298
                                                  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 =
 03299
                                                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 =
                                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<1»; };
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 // NOLINT
 03301
                                                                                     template<> struct ConwayPolynomial<3, 8> { using ZPZ = aerobus::zpz<3>; using type =
                                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>; // NOLINT template<> struct ConwayPolynomial<3, 9> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1>; }; //
 03302
                                                   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
                                                   template<> struct ConwayPolynomial<3, 11> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
                                                     ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                                                                 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<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<1
                                                                                     template<> struct ConwayPolynomial<3, 13> { using ZPZ = aerobus::zpz<3>; using type =
03306
                                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                     ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT</pre>
                                                                                        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<2>, ZPZV<1>,
                                                   ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<3, 15> { using ZPZ = aerobus::zpz<3>; using type =
03308
                                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                       ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                                                                   template<> struct ConwayPolynomial<3, 16> { using ZPZ = aerobus::zpz<3>; using type
                                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2
                                                   template<> struct ConwayPolynomial<3, 17> { using ZPZ = aerobus::zpz<3; using type = POLYV<ZPZV<1>, ZPZV<0>, ZP
03310
                                                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<1»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<3, 18> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                                   ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<3, 19> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1»; };</pre>
                                         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<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<1>,
                          ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2</pre>; };
                          // NOLINT
                                           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 =
03316
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
03317
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
03318
                                        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 template<> struct ConwayPolynomial<5, 6> { using ZPZ = aerobus::zpz<5>; using type =
03319
                        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>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
                                                                                                                                                                                                                                                                                                                                                                                                                                    // NOLINT
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<4>; };
03322
                                         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<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<3»; }; //
                         NOLINT
                                           template<> struct ConwayPolynomial<5, 10> { using ZPZ = aerobus::zpz<5>; using type =
                          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>
03324
                                           template<> struct ConwayPolynomial<5, 11> { using ZPZ = aerobus::zpz<5>; using type
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
03325
                                            template<> struct ConwayPolynomial<5, 12> { using ZPZ = aerobus::zpz<5>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, 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>, ZPZV<
03326
                          ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                                           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<1>, ZPZV<1>, ZPZV<0>, ZPZV<4>,
                           ZPZV<2>, ZPZV<3>, ZPZV<0>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
03328
                                         template<> struct ConwayPolynomial<5, 15> { using ZPZ = aerobus::zpz<5>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<3»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type =
03329
                           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>,
                           ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2»; }; // NOLINT
03330
                                         template<> struct ConwayPolynomial<5, 17> { using ZPZ = aerobus::zpz<5>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<3»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<5, 183 { using ZPZ = aerobus::zpz<5; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZP
03331
                          03332
                                         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<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<4>, ZPZV<3>, ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<0>, ZPZV<4>, ZPZV<1>, ZPZV<2>; };
                          // NOLINT
03334
                                          template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
                         POLYV<ZPZV<1>, ZPZV<4»; }; // NOLINT
03335
                                           template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
                         POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3»; }; // NOLINT
03336
                                         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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; }; // NOLINT
03337
                                          template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
03338
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT
                                          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
                                           template<> struct ConwayPolynomial<7, 7> { using ZPZ = aerobus::zpz<7>; using type =
03340
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
03341
                                           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<6>, ZPZV<5>, 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
                          NOLINT
03343
                         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<1>, ZPZV<2>, ZPZV<3>,
                          ZPZV<3»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<7, 11> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                          \text{ZPZV}<1>, \text{ZPZV}<4>; }; // NOLINT
                         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<4>, ZPZV<0>,
03345
```

```
ZPZV<5>, ZPZV<0>, ZPZV<3»; };
                                                       template<> struct ConwayPolynomial<7, 13> { using ZPZ = aerobus::zpz<7>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<4»; }; // NOLINT
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<6>, ZPZV<6 , ZPZV<6
03347
                                                        template<> struct ConwayPolynomial<7, 15> { using ZPZ = aerobus::zpz<7>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<6>, ZPZV<6>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<4»; }; // NOLINT

template<> struct ConwayPolynomial<7, 16> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
03349
                                 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<0>, ZPZV<4>; }; // NOLINT
template<> struct ConwayPolynomial<7, 18> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<5, ZPZV<6</pre>, ZPZV<1>,
                                  ZPZV<6>, ZPZV<5>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<2>, ZPZV<2>, ZPZV<3»; }; // NOLINT</pre>
                                                      template<> struct ConwayPolynomial<7, 19> { using ZPZ = aerobus::zpz<7>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  NOLINT
                                 \label{eq:convergence} template<> struct ConwayPolynomial<7, 20> \{ using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<6>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3 , ZPZ
03354
                                                       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 =
03355
                                POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2»; }; // NOLINT
 03356
                                                         template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
                                 POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<2>, ZPZY<2>, ZPZY<2>; // NOLINT template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type =
 03357
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<11, 5> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<0>, ZPZV<0>, ZPZV<10, ZPZV<10, ZPZV<10, ZPZV<10+; is ing ZPZ = aerobus::zpz<11; using type =
template<> struct ConwayPolynomial<11, 6> { using ZPZ = aerobus::zpz<11>; using type =
 03358
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<6>, ZPZV<7>, ZPZV<2»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<11, 7> { using ZPZ = aerobus::zpz<11>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<9»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<11, 8> { using ZPZ = aerobus::zpz<11>; using type =
03361
                                 POLYYCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<1>, ZPZV<1>, ZPZV<2); }; // NOLINT template<> struct ConwayPolynomial<11, 9> { using ZPZ = aerobus::zpz<11>; using type =
03362
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9
                                  NOLINT
03363
                                                    template<> struct ConwayPolynomial<11, 10> { using ZPZ = aerobus::zpz<11>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<10>, ZPZV<6>, ZPZV<6>,
                                  ZPZV<2»: }: // NOLINT
                                                        template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type
03364
                                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<10>, ZPZV<9»; }; // NOLINT</pre>
03365
                                                     template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<2>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<2»; }; // NOLINT
03366
                                                       template<> struct ConwayPolynomial<11, 13> { using ZPZ = aerobus::zpz<11>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<9»; }; // NOLINT</pre>
                                                      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<2>, ZPZV<9>, ZPZV<6>,
                                 ZPZV<4>, ZPZV<8>, ZPZV<6>, ZPZV<10>, ZPZV<0>, ZP
                                  ZPZV<7>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<0>, ZPZV<9»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                   // NOLINT
                                                    template<> struct ConwayPolynomial<11, 16> { 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<1>, ZPZV<1>, ZPZV<10>,
                                 ZPZV<1>, ZPZV<3>, ZPZV<5>, ZPZV<3>, ZPZV<10>, ZPZV<9>, ZPZV<2»; }; // NOLINT
    template<> struct ConwayPolynomial<11, 17> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03370
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<9»; }; // NOLINT</pre>
                                                        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<3>, ZPZV<8>, ZPZV<8>, ZPZV<10>, ZPZV<8>,
                                 ZPZV<3>, ZPZV<9>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<9>, ZPZV<8>, ZPZV<2>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<11, 19> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03372
                                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, 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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<4>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<2>; };
                                  // NOLINT
                                                        template<> struct ConwayPolynomial<13, 1> { using ZPZ = aerobus::zpz<13>; using type =
                                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
                                                      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 =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<13, 5> { using ZPZ = aerobus::zpz<13>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<11»; }; // NOLINT
                               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 =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<11»; };
                                                      template<> struct ConwayPolynomial<13, 8> { using ZPZ = aerobus::zpz<13>; using type =
 03381
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<12>, ZPZV<2>, ZPZV<3>, ZPZV<2»; };
 03382
                                                       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 =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5 , ZPZV<5
                                   ZPZV<2»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<13, 11> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11»; }; // NOLINT
                                                          template<> struct ConwayPolynomial<13, 12> { using ZPZ = aerobus::zpz<13>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<8>, ZPZV<11>, ZPZV<3>, ZPZV<1>, ZPZV<4>, ZPZV<4 , ZPZV<
                                                       template<> struct ConwayPolynomial<13, 13> { using ZPZ = aerobus::zpz<13>; using type =
03386
                                  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
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<6>,
                                   ZPZV<11>, ZPZV<7>, ZPZV<10>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<11>, ZPZV<11>, ZPZV<10>, ZPZV<11>, ZPZV<10>, ZPZV<11>; ZPZV<1
03388
03389
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   \texttt{ZPZV} < \texttt{8>, ZPZV} < \texttt{2>, ZPZV} < \texttt{12>, ZPZV} < \texttt{9>, ZPZV} < \texttt{12>, ZPZV} < \texttt{6>, ZPZV} < \texttt{2*; }; // \texttt{NOLINT} 
                                 template<> struct ConwayPolynomial<13, 17> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
 03390
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<6>, ZPZV<11»; }; // NOLINT
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<0>, ZPZV<0>, ZPZV<1>, ZPZV<10, ZPZV<1>, ZPZV<10>, ZPZV<10>, ZPZV<1>, ZPZV<10>, ZPZV<10>,
                                   ZPZV<11>, ZPZV<9>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<0>, ZPZV<9>, ZPZV<2»; }; // NOLINT</pre>
03392
                                                      template<> struct ConwayPolynomial<13, 19> { using ZPZ = aerobus::zpz<13>; using type =
                                 POLYV<ZPZV<0>, ZPZV<0>, ZPZV<0 , ZPZV<0
                                 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<1>, ZPZV<0>, Z
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    7.P7.V<12>.
                                  ZPZV<9>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<7>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<8>, ZPZV<11>, ZPZV<2»; };</pre>
                                   // NOLINT
03394
                                                         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 =
03395
                                  POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»; }; // NOLINT
 03396
                                                         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 =
03397
                                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
                                                          template<> struct ConwayPolynomial<17, 6> { using ZPZ = aerobus::zpz<17>; using type =
 03399
                                  \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 2>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 10>, \ \texttt{ZPZV} < 3>, \ \texttt{ZPZV} < 3>; \ \}; \ \ // \ \ \texttt{NOLINT} 
                               03400
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<12>, ZPZV<0>, ZPZV<6>, ZPZV<3»; };
03402
                                                      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<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<8>, ZPZV<14»; };
                                   // NOLINT
                                                         template<> struct ConwayPolynomial<17, 10> { using ZPZ = aerobus::zpz<17>; using type =
03403
                                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<12>,
                                  ZPZV<3»; }; // NOLINT</pre>
                                                         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
03405
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<13>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<14>, ZPZV<9>, ZPZV<9>, ZPZV<3»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<3>, ZPZV<3 , ZPZV<3
03407
                                  ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                                                       template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::zpz<17>; using type
                                 POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                                 ZPZV<4>, ZPZV<16>, ZPZV<6>, ZPZV<14>, ZPZV<14>, ZPZV<14»; }; // NOLINT
    template<> struct ConwayPolynomial<17, 16> { 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<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
 03409
```

```
ZPZV<5>, ZPZV<2>, ZPZV<12>, ZPZV<13>, ZPZV<12>, ZPZV<1>, ZPZV<3»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<17, 17> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
                                    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<1>, ZPZV<1>, ZPZV<0>, ZPZV<5, ZPZV<1</pre>
                                   ZPZV<7>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<11>, ZPZV<13>, ZPZV<13>, ZPZV<9>, ZPZV<3»; }; // NOLINT
                                                         template<> struct ConwayPolynomial<17, 19> { 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<11>, ZPZV<14*; }; //</pre>
                                   NOLINT
                                   template<> struct ConwayPolynomial<17, 20> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<15>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<15>, ZPZV<15>, ZPZV<15, Z
                                     ZPZV<3»; }; // NOLINT</pre>
                                                             template<> struct ConwayPolynomial<19, 1> { using ZPZ = aerobus::zpz<19>; using type =
                                  POLYV<ZPZV<1>, ZPZV<17»; }; // NOLINT
                                                          template<> struct ConwayPolynomial<19, 2> { using ZPZ = aerobus::zpz<19>; using type =
 03415
                                   POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2»; }; // NOLINT
                                                         template<> struct ConwayPolynomial<19, 3> { using ZPZ = aerobus::zpz<19>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type =
  03417
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 5> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17»; }; // NOLINT
  03418
                                                            template<> struct ConwayPolynomial<19, 6> { using ZPZ = aerobus::zpz<19>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<17>, ZPZV<6>, ZPZV<2»; }; // NOLINT
  03420
                                                         template<> struct ConwayPolynomial<19, 7> { using ZPZ = aerobus::zpz<19>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                                                        template<> struct ConwayPolynomial<19, 8> { using ZPZ = aerobus::zpz<19>; using type =
 03421
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<12, ZPZV<12, ZPZV<12, ZPZV<19, ZPZV<2, ZPZV<10, ZPZV<2, ZPZV<10, ZPZV<3, ZPZV
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<14>, ZPZV<16>, ZPZV<17»; };
                                    // 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>,
 03423
                                    ZPZV<2»; }; // NOLINT</pre>
                                                            template<> struct ConwayPolynomial<19, 11> { using ZPZ = aerobus::zpz<19>; using type
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    ZPZV<8>, ZPZV<17»; };</pre>
                                                                                                                                                                         // NOLINT
 03425
                                                        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<2>, ZPZV<2>, ZPZV<9>, ZPZV<9>, ZPZV<3>, ZPZV<2>, ZPZV<3>, ZPZV<2>, ZPZV<3>, ZPZV<3 , ZPZV<3
                                                          template<> struct ConwayPolynomial<19, 13> { using ZPZ = aerobus::zpz<19>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03427
                                                        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<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 15> { using ZPZ = aerobus::zpz<19>; using type =
 03428
                                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZ
                                    ZPZV<11>, ZPZV<13>, ZPZV<15>, ZPZV<14>, ZPZV<0>, ZPZV<17»; }; // NOLINT</pre>
 03429
                                                        template<> struct ConwayPolynomial<19, 16> { using ZPZ = aerobus::zpz<19>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
 03430
                                    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>, ZPŽV<0>, ZPZV<0>, ZPZV<0
                                   ZPZV<17>, ZPZV<5>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<15, ZPZV<3>, ZPZV<3>, ZPZV<14>, ZPZV<14>, ZPZV<2*, }; // NOLINT template<> struct ConwayPolynomial<19, 19> { 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<0>, ZPZV<0>, ZPZV<17»; }; //</pre>
 03433
                                                       template<> struct ConwayPolynomial<19, 20> { using ZPZ = aerobus::zpz<19>; using type =
                                   POLYV-ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<16>, ZPZV<18-, ZPZV<18-, ZPZV<18-, ZPZV<19-, ZPZV<
                                    }; // NOLINT
 03434
                                                              template<> struct ConwayPolynomial<23, 1> { using ZPZ = aerobus::zpz<23>; using type =
                                   POLYV<ZPZV<1>, ZPZV<18»; }; // NOLINT
  03435
                                                       template<> struct ConwayPolynomial<23, 2> { using ZPZ = aerobus::zpz<23>; using type =
                                  POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; }; // NOLINT
                                                         template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
  03436
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; }; // NOLINT template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT
  03438
                                                        template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<18»; }; // NOLINT
  03439
                                                         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
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<18»; }; // NOLINT
  03441
                                                        template<> struct ConwayPolynomial<23, 8> { using ZPZ = aerobus::zpz<23>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5»; };
                                   template<> struct ConwayPolynomial<23, 9> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<9>, ZPZV<18»; };</pre>
```

```
// NOLINT
03443
                                                  template<> struct ConwayPolynomial<23, 10> { using ZPZ = aerobus::zpz<23>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<1>,
                              ZPZV<5»; }; // NOLINT</pre>
                                                  template<> struct ConwayPolynomial<23, 11> { 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<2>, ZPZV<18»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<23, 12> { using ZPZ = aerobus::zpz<23>; using type
                             POLYYCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<21>, ZPZV<21>, ZPZV<15>, ZPZV<14>, ZPZV<12>, ZPZV<15, ZPZV<15, ZPZV<15, ZPZV<16, ZPZV<18, 
03446
                                               template<> struct ConwayPolynomial<23, 13> { using ZPZ = aerobus::zpz<23>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<18»; };</pre>
                                                                                                                                                                                                                                          // NOLINT
                                                  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<1>, ZPZV<5>, ZPZV<16>, ZPZV<1>,
                             ZPZV<18>, ZPZV<19>, ZPZV<1>, ZPZV<22>, 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<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3 , ZPZV<3
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<19>, ZPZV<16>, ZPZV<13>, ZPZV<1>, ZPZV<14>, ZPZV<17>, ZPZV<5>; }; // NOLINT
    template<> struct ConwayPolynomial<23, 17> { using ZPZ = aerobus::zpz<23>; using type =
03450
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<20>; zPZV
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<2>, ZPZV<1>,
                               \texttt{ZPZV} < 18 >, \ \texttt{ZPZV} < 3 >, \ \texttt{ZPZV} < 16 >, \ \texttt{ZPZV} < 21 >, \ \texttt{ZPZV} < 0 >, \ \texttt{ZPZV} < 11 >, \ \texttt{ZPZV} < 3 >, \ \texttt{ZPZV} < 19 >, \ \texttt{ZPZV} < 5 »; \ \}; \ \ // \ \texttt{NOLINT} 
                             template<> struct ConwayPolynomial<23, 19> { 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>, ZPZV<5>, ZPZV<5>, ZPZV<18»; }; //</pre>
                              NOLINT
                                                  template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type 
                             POLYV<ZPZV<1>, ZPZV<27»; }; // NOLINT
03454
                                                  template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
                             POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
03455
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<27»; }; // NOLINT
03456
                                                  template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<2»; }; // NOLINT
03457
                                               template<> struct ConwayPolynomial<29, 5> { using ZPZ = aerobus::zpz<29>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<27»; }; // NOLINT template<> struct ConwayPolynomial<29, 6> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<17>, ZPZV<13>, ZPZV<2»; }; // NOLINT
03458
                                                  template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<27»; };
03460
                                              template<> struct ConwayPolynomial<29, 8> { using ZPZ = aerobus::zpz<29>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<2»; }; //
                             NOLINT
                                                template<> struct ConwayPolynomial<29, 9> { using ZPZ = aerobus::zpz<29>; using type =
03461
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<22>, ZPZV<22, ZPZV<27»; };
03462
                                                template<> struct ConwayPolynomial<29, 10> { using ZPZ = aerobus::zpz<29>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<8>, ZPZV<17>, ZPZV<22>, ZPZV<22>, ZPZV<2»; }; // NOLINT
03463
                                                  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<8>, ZPZV<27»; };</pre>
                                                                                                                                                 // 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<19>, ZPZV<28>, ZPZV<28>, ZPZV<26>, ZPZV<25>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>; }; // NOLINT
                                                template<> struct ConwayPolynomial<29, 13> { using ZPZ = aerobus::zpz<29>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<27»; };</pre>
                                                                                                                                                                                                                                          // NOLINT
                                              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
template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::zpz<29>; using type =
03467
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>,
                              ZPZV<14>, ZPZV<8>, ZPZV<1>, ZPZV<12>, ZPZV<26>, ZPZV<27»; }; // NOLINT</pre>
                                                  template<> struct ConwayPolynomial<29, 16> { using ZPZ = aerobus::zpz<29>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<2>, ZPZV<18>, ZPZV<23>, ZPZV<1>, ZPZV<27>, ZPZV<10>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<29, 17> { using ZPZ = aerobus::zpz<29>; using type =
03469
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<27»; }; // NOLINT</pre>
                                              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<2+>, ZPZV<24>, ZPZV<1>
                             ZPZV<6>, ZPZV<26>, ZPZV<2>, ZPZV<10>, ZPZV<8>, ZPZV<16>, ZPZV<19>, ZPZV<14>, ZPZV<14>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<29, 19> { using ZPZ = aerobus::zpz<29>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZP
03471
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<27»; }; //</pre>
03472
                                                template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<28»; }; // NOLINT 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 =
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; // NOLINT template<> struct ConwayPolynomial<31, 4> { using ZPZ = aerobus::zpz<31>; using type =
03475
                                    POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<31, 5> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; }; // NOLINT
template<> struct ConwayPolynomial<31, 6> { using ZPZ = aerobus::zpz<31>; using type =
 03476
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<16>, ZPZV<8>, ZPZV<3»; };
 03478
                                                              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<0>, ZPZV<1>, ZPZV<28»; };
03479
                                                               template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<12>, ZPZV<24>, ZPZV<3»; };
                                                                   template<> struct ConwayPolynomial<31, 9> { using ZPZ = aerobus::zpz<31>; using type =
                                      POLYV<ZPZV<1>, ZPZV<0>, 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<3»; }; // NOLINT</pre>
                                                                   template<> struct ConwayPolynomial<31, 11> { using ZPZ = aerobus::zpz<31>; using type
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<20>, ZPZV<28»; }; // NOLINT
  template<> struct ConwayPolynomial<31, 12> { using ZPZ = aerobus::zpz<31>; using type =
03483
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<28>, ZPZV<2>, ZPZV<2>, ZPZV<5>, ZPZV<5>, ZPZV<14>, 
                                        ZPZV<25>, ZPZV<12>, ZPZV<3»; }; // NOLINT</pre>
                                                                   template<> struct ConwayPolynomial<31, 13> { using ZPZ = aerobus::zpz<31>; using type
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<28»; }; // NOLINT</pre>
                                      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<1>, ZPZV<18>, ZPZV<18>, ZPZV<6>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type =
03485
03486
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         \mbox{ZPZV<29>, ZPZV<12>, ZPZV<13>, ZPZV<23>, ZPZV<25>, ZPZV<28*; }; \ // \ \mbox{NOLINT} 
                                      template<> struct ConwayPolynomial<31, 16> \{ using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
 03487
                                      ZPZV<24>, ZPZV<26>, ZPZV<28>, ZPZV<11>, ZPZV<19>, ZPZV<27>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<31, 17> { 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<10>, ZPZV<28»; }; // NOLINT</pre>
03489
                                                              template<> struct ConwayPolynomial<31, 18> { using ZPZ = aerobus::zpz<31>; using type =
                                      POLYY<ZPZV<1>, 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<3
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<0>, ZPZV<0</pre>; };
03491
                                                                template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
                                      POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                                                  template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
 03492
                                      POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2»; }; // NOLINT
                                                                   template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT
template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
 03494
                                     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
03495
                                                               template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
 03496
                                    templates struct ConwayPolynomials37, 65 { using ZPZ = aerobus::ZpZs37; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, 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<0>, ZPZV<7>, ZPZV<35»; }; // NOLINT template<> struct ConwayPolynomial<37, 8> { using ZPZ = aerobus::zpZ<37>; using type =
 03497
03498
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<20>, ZPZV<27>, ZPZV<27>, ZPZV<1>, ZPZV<2, };
                                                                  template<> struct ConwayPolynomial<37, 9> { using ZPZ = aerobus::zpz<37>; using type
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<20, ZPZV<32, ZPZV<35»; };
                                        // NOLINT
03500
                                      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<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»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>
 03502
                                        ZPZV<18>, ZPZV<33>, ZPZV<2»; }; // NOLINT</pre>
                                                                   template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type
                                        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<4>, ZPZV<35>, ZPZV<35>, ZPZV<1>, ZPZV<32>, ZPZV<16>, ZPZV<1>, ZPZV<29>, ZPZV<2»; }; // NOLINT
                                                                   template<> struct ConwayPolynomial<37, 15>
                                                                                                                                                                                                                                                                                                                                                            { 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<3>, ZPZV<31>, ZPZV<28>, ZPZV<27>, ZPZV<13>, ZPZV<34>, ZPZV<34>, ZPZV<35»; }; // NOLINT
                                      template<> struct ConwayPolynomial<37, 17> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
```

```
template<> struct ConwayPolynomial<37, 18> { using ZPZ = aerobus::zpz<37>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2
                                       template<> struct ConwayPolynomial<37, 19> { using ZPZ = aerobus::zpz<37>; using type =
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<23>, ZPZV<35»; }; //</pre>
                         NOLINT
03509
                                           template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type
                         POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                         template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
03510
                         POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6»; }; // NOLINT
                                         template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
03511
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<25»; }; // 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
03513
                                         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
                         template<> struct ConwayPolynomial<41, 6> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<39>, ZPZV<6>, ZPZV<6»; }; // NOLINT
03514
                                          template<> struct ConwayPolynomial<41, 7> { using ZPZ = aerobus::zpz<41>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<35»; }; //
03516
                                          template<> struct ConwayPolynomial<41, 8> { using ZPZ = aerobus::zpz<41>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<32>, ZPZV<20>, ZPZV<6>, ZPZV<6»; }; // NOLINT
                         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<35>, ZPZV<35»; };</pre>
03517
                         // NOLINT
                                         template<> struct ConwayPolynomial<41, 10> { using ZPZ = aerobus::zpz<41>; using type =
03518
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<8>, ZPZV<20>, ZPZV<30>,
                         ZPZV<6»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<41, 11> { using ZPZ = aerobus::zpz<41>; using type =
03519
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<20>, ZPZV<35»; };</pre>
                                                                                                                              // NOLINT
                                          template<> struct ConwayPolynomial<41, 12> { using ZPZ = aerobus::zpz<41>; using type
03520
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26>, ZPZV<13>, ZPZV<34>, ZPZV<24>,
                          ZPZV<21>, ZPZV<27>, ZPZV<6»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<41, 13> { using ZPZ = aerobus::zpz<41>; using type
03521
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            template<> struct ConwayPolynomial<41, 14> { using ZPZ = aerobus::zpz<41>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<15>, ZPZV<15>, 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>,
03523
                         ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<41, 17> { using ZPZ = aerobus::zpz<41>; using type
                         POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<0
                         template<> struct ConwayPolynomial<41, 18> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<6>, ZPZV<6>; // NOLINT template<> struct ConwayPolynomial<41, 19> { 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<0>, ZPZV<0>, ZPZV<10>, ZPZV<35»; }; //</pre>
                         NOLINT
03527
                                          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 =
                         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 =
03530
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3>; I // NOLINT
template<> struct ConwayPolynomial<43, 5> { using ZPZ = aerobus::zpz<43>; using type =
03531
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<40»; }; // NOLINT
03532
                                         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
03533
                        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<40»; }; // NOLINT
                                         template<> struct ConwayPolynomial<43, 8> { using ZPZ = aerobus::zpz<43>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<39>, ZPZV<20>, ZPZV<24>, ZPZV<3»; };
03535
                                       template<> struct ConwayPolynomial<43, 9> { using ZPZ = aerobus::zpz<43>; using type =
                         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 =
03536
                         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>
03537
                                         template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40»; }; // NOLINT
                                          template<> struct ConwayPolynomial<43, 12> { using ZPZ = aerobus::zpz<43>; using type =
03538
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<27>, ZPZV<16>, ZPZV<17>, ZPZV<6>, ZPZV<6>, ZPZV<28, ZPZV<28, ZPZV<28, ZPZV<38, ZP
                                        template<> struct ConwayPolynomial<43, 13> { using ZPZ = aerobus::zpz<43>; using type =
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
03540
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<38>, ZPZV<22>, ZPZV<24>,
                            ZPZV<37>, ZPZV<18>, ZPZV<4), ZPZV<19>, ZPZV<3»; }; // 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<37>,
                            ZPZV<22>, ZPZV<42>, ZPZV<4>, ZPZV<15>, ZPZV<37>, ZPZV<40»; }; // NOLINT
template<> struct ConwayPolynomial<43, 17> { using ZPZ = aerobus::zpz<43>; using type =
03542
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<40»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<43, 18> { using ZPZ = aerobus::zpz<43>; using type =
03543
                            Template<> struct ConwayFolynomial
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<34>, ZPZV<34|, ZPZV<34>, ZPZV<35, ZPZ
03544
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40»; }; //</pre>
                             NOLINT
03545
                                                 template<> struct ConwayPolynomial<47, 1> { using ZPZ = aerobus::zpz<47>; using type =
                            POLYV<ZPZV<1>, ZPZV<42»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<47, 2> { using ZPZ = aerobus::zpz<47>; using type =
03546
                             POLYV<ZPZV<1>, ZPZV<45>, ZPZV<5»; }; // NOLINT
                                                template<> struct ConwayPolynomial<47, 3> { using ZPZ = aerobus::zpz<47>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42»; }; // NOLINT template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type =
03548
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<47, 5> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT
03549
                                                  template<> struct ConwayPolynomial<47, 6> { using ZPZ = aerobus::zpz<47>; using type =
                            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 =
03551
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT
03552
                                              template<> struct ConwayPolynomial<47, 8> { using ZPZ = aerobus::zpz<47>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3>, ZPZV<5; // NOLINT template<> struct ConwayPolynomial<47, 9> { using ZPZ = aerobus::zpz<47>; using type =
03553
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<42»; };
                              // 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>,
03554
                             ZPZV<5»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<47, 11> { using ZPZ = aerobus::zpz<47>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<6>, ZPZV<42»; };</pre>
                                                                                                                                            // NOLINT
03556
                                              template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<46 , ZPZV<47 , ZPZV
03557
                                                 template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03558
                                              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<50>, ZPZV<50>, ZPZV<32>, ZPZV<50>, ZPZV<50
03559
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<43>,
                             ZPZV<31>, ZPZV<14>, ZPZV<42>, ZPZV<13>, ZPZV<17>, ZPZV<42»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<47, 17> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03560
                            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<6>, ZPZV<6>, ZPZV<41>, ZPZV<41>, ZPZV<41>, ZPZV<41>, ZPZV<42>,
03561
                             ZPZV<26>, ZPZV<44>, ZPZV<24>, ZPZV<22>, ZPZV<11>, ZPZV<5>, ZPZV<45>, ZPZV<33>, ZPZV<5»; }; // NOLINT</pre>
                            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<35>, ZPZV<42»; }; //</pre>
                             NOLINT
03563
                                                 template<> struct ConwayPolynomial<53, 1> { using ZPZ = aerobus::zpz<53>; using type =
                            POLYV<ZPZV<1>, ZPZV<51»; }; // NOLINT
03564
                                              template<> struct ConwayPolynomial<53, 2> { using ZPZ = aerobus::zpz<53>; using type =
                            POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2»; }; // NOLINT
03565
                                                 template<> struct ConwayPolynomial<53, 3> { using ZPZ = aerobus::zpz<53>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
03566
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<38>, ZPZV<2»; }; // NOLINT
                                                template<> struct ConwayPolynomial<53, 5> { using ZPZ = aerobus::zpz<53>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT
03568
                                                 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
03569
                            POLYV<2P2V<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<51»; }; //
                                              template<> struct ConwayPolynomial<53, 8> { using ZPZ = aerobus::zpz<53>; using type =
03570
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<29>, ZPZV<18>, ZPZV<1>, ZPZV<2»; };
03571
                                              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<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<1>, ZPZV<1>, ZPZV<2>, ZPZV<27>, ZPZV<25>, ZPZV<29>,
                             ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<53, 11> { using ZPZ = aerobus::zpz<53>; using type =
                             \texttt{POLYV} < \texttt{ZPZV} < \texttt{1}>, \ \texttt{ZPZV} < \texttt{0}>, \
                              ZPZV<15>, ZPZV<51»; }; // NOLINT</pre>
```

```
template<> struct ConwayPolynomial<53, 12> { using ZPZ = aerobus::zpz<53>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<34>, ZPZV<13>, ZPZV<10>, ZPZV<42>, ZPZV<34>, ZPZV<13>, ZPZV<10>, ZPZV<42>, ZPZV<34>, ZPZV<13>, ZPZV<10>, ZPZV<41>, ZPZV
                                      template<> struct ConwayPolynomial<53, 13> { using ZPZ = aerobus::zpz<53>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<52>, ZPZV<28>, ZPZV<28), ZPZV<51»; }; // NOLINT template<> struct ConwayPolynomial<53, 14> { using ZPZ = aerobus::zpz<53>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, 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<2>, 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 =
03577
03578
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          \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<0>, ZPZV<5>, ZPZV<5-, ZPZV<5-,
03579
                        ZPZV<27, ZPZV<28>, ZPZV<39>, ZPZV<39>, ZPZV<46>, ZPZV<85, ZPZV<61, ZPZV<25; ; // NOLINT template<> struct ConwayPolynomial<53, 19> { using ZPZ = aerobus::zpz<53>; 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<11>, ZPZV<51»; }; //</pre>
                         NOLINT
03581
                                        template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
                        POLYV<ZPZV<1>, ZPZV<57»; }; // NOLINT
03582
                                         template<> struct ConwayPolynomial<59, 2> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2»; }; // NOLINT
                                       template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
03583
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; }; // NOLINT template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
03584
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<2»; }; // NOLINT

template<> struct ConwayPolynomial<59, 5> { using ZPZ = aerobus::zpz<59>; using type =
03585
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<57»; }; // NOLINT
                                         template<> struct ConwayPolynomial<59, 6> { using ZPZ = aerobus::zpz<59>; using type =
                        03587
                                         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<1>, ZPZV<57»; }; // NOLINT
                                        template<> struct ConwayPolynomial<59, 8> { using ZPZ = aerobus::zpz<59>; using type =
03588
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<32>, ZPZV<2>, ZPZV<50>, ZPZV<50; };
                         NOLINT
                                       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<0>, ZPZV<32>, ZPZV<32>, ZPZV<47>, ZPZV<57»; };
                         // NOLINT
                                       template<> struct ConwayPolynomial<59, 10> { using ZPZ = aerobus::zpz<59>; using type =
03590
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28>, ZPZV<25>, ZPZV<4>, ZPZV<39>, ZPZV<15>,
                         ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<59, 11> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<6>, ZPZV<57»; }; // NOLINT
    template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type =
03592
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<21>, ZPZV<38>,
                         ZPZV<8>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<59, 13> { using ZPZ = aerobus::zpz<59>; using type =
                         \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ 
                        ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57»; }; // NOLINT
template<> struct ConwayPolynomial<59, 14> { using ZPZ = aerobus::zpz<59>; using type =
03594
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<51>, ZPZV<11>,
                         ZPZV<13>, ZPZV<25>, ZPZV<32>, ZPZV<26>, ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<24>, ZPZV<23>, ZPZV<13>, ZPZV<39>, ZPZV<58>, ZPZV<57»; }; // NOLINT
    template<> struct ConwayPolynomial<59, 17> { using ZPZ = aerobus::zpz<59>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03596
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<57»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<37>, ZPZV<37>, ZPZV<38>, ZPZV<27>,
                        ZPZV<11>, ZPZV<14>, ZPZV<7>, ZPZV<44>, ZPZV<16>, ZPZV<47>, ZPZV<34>, ZPZV<32>, ZPZV<2»; }; // NOLINT
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>,
03598
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<57»; }; //</pre>
                         NOLINT
                                          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 =
03600
                        POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»: }; // NOLINT
                                         template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
03601
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; }; // NOLINT
                                       template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
03602
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2»; }; // NOLINT
                       template<> struct ConwayPolynomial<61, 5> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<59»; }; // NOLINT
03603
                                         template<> struct ConwayPolynomial<61, 6> { using ZPZ = aerobus::zpz<61>; using type =
03604
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<3>, ZPZV<29>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<61, 7> { using ZPZ = aerobus::zpz<61>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<59»; };
                      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»; }; // NOLINT
template<> struct ConwayPolynomial<61, 9> { using ZPZ = aerobus::zpz<61>; using type =
03606
03607
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<50>, ZPZV<59»; };
03608
                                             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<66>,
                             ZPZV<2»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<61, 11> { using ZPZ = aerobus::zpz<61>; using type =
03609
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                 // NOLINT
                             ZPZV<18>, ZPZV<59»; };</pre>
                                              template<> struct ConwayPolynomial<61, 12> { using ZPZ = aerobus::zpz<61>; using type
03610
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<42>, ZPZV<33>, ZPZV<8>, ZPZV<38>, ZPZV<14>, ZPZV<15>, ZPZV<2»; }; // NOLINT
                                               template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type =
03611
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<59»; };</pre>
                                             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<1>, ZPZV<48>, ZPZV<48>, ZPZV<26>, ZPZV<11>, ZPZV<8>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<26>, ZPZV<11>, ZPZV<8>, ZPZV<80>, ZPZV<10>, 
03613
                             ZPZV<35>, ZPZV<44>, ZPZV<25>, ZPZV<23>, ZPZV<51>, ZPZV<59»; }; // NOLINT
                            template<> struct ConwayPolynomialc61, 17> { 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<10>, ZPZV<59»; }; // NOLINT</pre>
                            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<3>, ZPZV<3>, ZPZV<35>, ZPZV<36>, ZPZV<13>,
03615
                            ZPZV<36>, ZPZV<4>, ZPZV<32>, ZPZV<57>, ZPZV<42>, ZPZV<25>, ZPZV<25>, ZPZV<25>, ZPZV<52>, ZPZV<62>; }; // NOLINT
template<> struct ConwayPolynomial<61, 19> { using ZPZ = aerobus::zpz<61>; using type =
                                                                                                                                                                                                                                                                                                                                                                                                                                                                    7.P7.V<0>,
                              \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ 
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; };</pre>
                             NOLINT
                                               template<> struct ConwayPolynomial<67, 1> { using ZPZ = aerobus::zpz<67>; using type =
03617
                           POLYV<ZPZV<1>, ZPZV<65»; }; // NOLINT
                                                template<> struct ConwayPolynomial<67, 2> { using ZPZ = aerobus::zpz<67>; using type =
                            POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
                                                template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
03619
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65»; }; // NOLINT

template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<54>, ZPZV<2»; }; // NOLINT
03620
                                                template<> struct ConwayPolynomial<67, 5> { using ZPZ = aerobus::zpz<67>; using type =
03621
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<65»; }; // NOLINT
03622
                                              template<> struct ConwayPolynomial<67, 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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5 , ZPZV<
03623
                                                template<> struct ConwayPolynomial<67, 8> { using ZPZ = aerobus::zpz<67>; using type :
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<46>, ZPZV<46>, ZPZV<46+, ZPZV<64>, ZPZV<2»; };
                            template<> struct ConwayPolynomial<67, 9> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<55>, ZPZV<55>; ZPZV<65»; };
                             // NOLINT
03626
                                                template<> struct ConwayPolynomial<67, 10> { using ZPZ = aerobus::zpz<67>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<0>, ZPZV<16>, ZPZV<7>, ZPZV<23>,
                             ZPZV<2»; }; // NOLINT</pre>
03627
                                                template<> struct ConwayPolynomial<67, 11> { 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<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<65»; // NOLINT
                                                 template<> struct ConwayPolynomial<67, 12> { using ZPZ = aerobus::zpz<67>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<57>, ZPZV<27>, ZPZV<4>, ZPZV<45, ZPZV<64>,
                              ZPZV<21>, ZPZV<27>, ZPZV<2»; }; // NOLINT</pre>
03629
                                                 template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                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<17>, ZPZV<22>, ZPZV<5>,
                             ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<2»; };</pre>
                                                                                                                                                                                                                                                                                    // NOLINT
03631
                                             template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
03632
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<65»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<67, 18> { using ZPZ = aerobus::zpz<67>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, 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
03634
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<18>, ZPZV<65»; };</pre>
                                                template<> struct ConwayPolynomial<71, 1> { using ZPZ = aerobus::zpz<71>; using type =
03635
                           POLYV<ZPZV<1>, ZPZV<64»; }; // NOLINT
                                                template<> struct ConwayPolynomial<71, 2> { using ZPZ = aerobus::zpz<71>; using type =
03636
                            POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7»; }; // NOLINT
                                                template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; };
                                                                                                                                                                                                                                                                  // NOLINT
03638
                                             template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, 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<18>, ZPZV<64»; };
                              template<> struct ConwayPolynomial<71, 6> { using ZPZ = aerobus::zpz<71>; using type = 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 =
 03641
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<64»; }; // NOLINT
03642
                                                    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»; };
                                                  template<> struct ConwayPolynomial<71, 9> { using ZPZ = aerobus::zpz<71>; using type =
03643
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43>, ZPZV<43>, ZPZV<64»; };
                                 // NOLINT
                                                   template<> struct ConwayPolynomial<71, 10> { using ZPZ = aerobus::zpz<71>; using type =
03644
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17>, ZPZV<26>, ZPZV<15, 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
                                                   template<> struct ConwayPolynomial<71, 12> { using ZPZ = aerobus::zpz<71>; using type =
03646
                                POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<22>, ZPZV<22>, ZPZV<25>, ZPZV<21>, ZPZV<28>, ZPZV<29>, ZPZV<21>, ZPZV<28>, ZPZV<29>, ZPZV<21>, ZPZV<28>, ZPZV<28 , ZPZV<2
                                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
  template<> struct ConwayPolynomial<71, 15> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
03648
                                ZPZV<32>, ZPZV<18>, ZPZV<52>, ZPZV<67>, ZPZV<49>, ZPZV<64*; }; // NOLINT
    template<> struct ConwayPolynomial<71, 17> { using ZPZ = aerobus::zpz<71>; using type =
03649
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 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<0 , ZPZV<0 ,
03650
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<64»; };</pre>
                                                   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 =
03652
                                POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
                                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»; }; // NOLINT template<> struct ConwayPolynomial<73, 5> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68»; }; // NOLINT
 03655
 03656
                                                   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 =
 03657
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<68»; }; // NOLINT
 03658
                                                 template<> struct ConwayPolynomial<73, 8> { using ZPZ = aerobus::zpz<73>; using type =
                                 POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<5>, ZPZV<39>, ZPZV<18>, ZPZV<5»; };
                                NOLINT
03659
                                                    template<> struct ConwayPolynomial<73, 9> { using ZPZ = aerobus::zpz<73>; using type
                                 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»; };
                                 // NOLINT
03660
                                                     template<> struct ConwayPolynomial<73, 10> { using ZPZ = aerobus::zpz<73>; using type =
                                 POLYV<2PZV<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
                                 ZPZV<5>, ZPZV<68»; }; // NOLINT</pre>
03662
                                                      template<> struct ConwayPolynomial<73, 12> { using ZPZ = aerobus::zpz<73>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<52>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<46>, ZPZV<26>, ZPZV<26 >, ZPZV<26 
03663
                                                     template<> struct ConwayPolynomial<73, 13> { using ZPZ = aerobus::zpz<73>; using type =
                                 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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03664
03665
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<68»; };</pre>
03666
                                                 template<> struct ConwayPolynomial<73, 19> { using ZPZ = aerobus::zpz<73>; using type
                                POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 NOLINT
                                                     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
                                                    template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
 03669
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76»; }; // NOLINT template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
 03670
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<79, 5> { using ZPZ = aerobus::zpz<79>; using type =
 03671
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<76»; }; // NOLINT 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
 03672
```

```
template<> struct ConwayPolynomial<79, 7> { using ZPZ = aerobus::zpz<79>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>; ; // NOLINT template<> struct ConwayPolynomial<79, 8> { using ZPZ = aerobus::zpz<79>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<60>, ZPZV<59>, ZPZV<48>, ZPZV<3»; };
                          NOLINT
                                         template<> struct ConwayPolynomial<79, 9> { using ZPZ = aerobus::zpz<79>; using type =
03675
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<57>, ZPZV<57>, ZPZV<19>, ZPZV<76»; };
                                         template<> struct ConwayPolynomial<79, 10> { using ZPZ = aerobus::zpz<79>; using type =
03676
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<51>, ZPZV<1>, ZPZV<30>, ZPZV<42>,
                          ZPZV<3»; }; // NOLINT
                                         template<> struct ConwayPolynomial<79, 11> { using ZPZ = aerobus::zpz<79>; using type =
03677
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        template<> struct ConwayPolynomial<79, 12> { using ZPZ = aerobus::zpz<79>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<45>, ZPZV<52>, ZPZV<50>, ZPZV<40>, ZPZV<40>, ZPZV<59>, ZPZV<50>, ZPZV<50>, ZPZV<3»; }; // NOLINT
03679
                                         template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          template<> struct ConwayPolynomial
79</pr>
19> { using ZPZ = aerobus::zpz<79>; using type
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0</pre>
, ZPZV<0
03681
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<76»; }; //</pre>
03682
                                          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 =
03683
                         POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; }; // NOLINT
03684
                                            template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
                         POLYVCZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT template<> struct ConwayPolynomial<83, 4> { using ZPZ = aerobus::zpz<83>; using type =
03685
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<83, 5> { using ZPZ = aerobus::zpz<83>; using type =
03686
                         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
                        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»; };</pre>
03689
                         template<> struct ConwayPolynomial<83, 9> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<81»; };</pre>
                         template<> struct ConwayPolynomial<83, 10> { using ZPZ = aerobus::zpz<83>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<73>, ZPZV<5>, ZPZV<5-, ZPZV<5-
03691
                          ZPZV<2»: }: // NOLINT</pre>
                                           template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type
03692
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<17>, ZPZV<81»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<12>, ZPZV<31>, ZPZV<31>, ZPZV<19>, ZPZV<65>,
03693
                          ZPZV<55>, ZPZV<75>, ZPZV<2»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<81»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<83, 17> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
03695
03696
                                          template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; 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<47>, ZPZV<81»; };</pre>
                          NOLINT
03697
                                           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 =
03698
                         POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; }; // NOLINT
                                           template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<86»; }; // NOLINT template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
03700
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<89, 5> { using ZPZ = aerobus::zpz<89>; using type =
03701
                         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 =
03702
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<80>, ZPZV<15>, ZPZV<3»; }; // NOLINT
03703
                                         template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type =
                         POLYV<ZPZV<1>, 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 =
03704
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<40>, ZPZV<79>, ZPZV<3»; }; //
03705
                                         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<52, ZPZV<62, ZPZV<66, ZPZV<86%; };
                          // NOLINT
03706
                                        template<> struct ConwavPolynomial<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<82>, ZPZV<42>,
                          ZPZV<3»; }; // NOLINT
                                         template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<88>,
                          ZPZV<26>, ZPZV<86»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<89, 12> { using ZPZ = aerobus::zpz<89>; using type =
03708
                          POLYY<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<5>, ZPZV<4>, ZPZV<5>, ZPZV<5-, ZPZV<5
                                         template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         template<> struct ConwayPolynomial<89, 17> { using ZPZ = aerobus::zpz<89>; using type
03710
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<86»; }; // NOLINT</pre>
03711
                                         template<> struct ConwayPolynomial<89, 19> { using ZPZ = aerobus::zpz<89>; using type
                                                                                                                                                                                                                                                                                                                                                                                                                        ZPZV<0>,
                          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<34>, ZPZV<86»; }; //
                          NOLINT
                                            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 =
                          POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5»; }; // NOLINT
                                         template<> struct ConwayPolynomial<97, 3> { using ZPZ = aerobus::zpz<97>; using type =
03714
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<92»; }; // NOLINT template<> struct ConwayPolynomial<97, 4> { using ZPZ = aerobus::zpz<97>; using type =
03715
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; }; // NOLINT
03716
                                         template<> struct ConwayPolynomial<97, 5> { using ZPZ = aerobus::zpz<97>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<92»; }; // NOLINT
03717
                                            template<> struct ConwayPolynomial<97, 6> { using ZPZ = aerobus::zpz<97>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<92>, ZPZV<58>, ZPZV<88>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<97, 7> { using ZPZ = aerobus::zpz<97>; using type =
03718
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<92»; };
                                           template<> struct ConwayPolynomial<97, 8> { using ZPZ = aerobus::zpz<97>; using type =
                         POLYV<ZPZV<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<1>, ZPZV<59>, ZPZV<59>, ZPZV<92»; };</pre>
03720
                           // NOLINT
                                            template<> struct ConwayPolynomial<97, 10> { using ZPZ = aerobus::zpz<97>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<20>,
                          ZPZV<5»; }; // NOLINT</pre>
03722
                                         template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<5>, ZPZV<92»; }; // NOLINT
   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<86>, ZPZV<78>, ZPZV<94>, ZPZV<59; }; // NOLINT
03724
                                         template<> struct ConwayPolynomial<97, 13> { using ZPZ = aerobus::zpz<97>; using type =
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            template<> struct ConwayPolynomial<97, 17> { using ZPZ = aerobus::zpz<97>; using type
03725
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          03726
                                         template<> struct ConwayPolynomial<97, 19> { using ZPZ = aerobus::zpz<97>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>
                          NOLINT
                                            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
                                         template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
03729
                       Templates struct Conwaysorynomialstor, 35 ( using ZPZ = aerobus:.2p2<1017, using type = POLYV<2p2V<1>, ZPZV<0>, ZPZV<3>, ZPZV<3), ZPZV<39,; }; // NOLINT template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
03730
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<78>, ZPZV<2»; }; // NOLINT
03731
                                         template<> struct ConwayPolynomial<101, 5> { using ZPZ = aerobus::zpz<101>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<99»; }; // NOLINT
                        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 =
03732
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<99»; };
                                          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»; }; //
                          NOLINT
03735
                                          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<0>, ZPZV<64>, ZPZV<47>, ZPZV<99»; };
                                         template<> struct ConwayPolynomial<101, 10> { using ZPZ = aerobus::zpz<101>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<4>, ZPZV<49>, ZPZV<100>, ZPZV<100>, ZPZV<52>,
                          ZPZV<2»; }; // NOLINT</pre>
03737
                                         template<> struct ConwayPolynomial<101, 11> { using ZPZ = aerobus::zpz<101>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<31>, ZPZV<99»; };</pre>
                                                                                                                                    // NOLINT
                                         template<> struct ConwayPolynomial<101, 12> { using ZPZ = aerobus::zpz<101>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<79>, ZPZV<64>, ZPZV<39>, ZPZV<78>, ZPZV<48>,
                          ZPZV<84>, ZPZV<21>, ZPZV<2»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<101, 13> { using ZPZ = aerobus::zpz<101>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03739
```

```
ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<99»; };</pre>
                                            template<> struct ConwayPolynomial<101, 17> { 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<31>, ZPZV<99»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<101, 19> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2+, ZPZV<29»; }; //</pre>
                                            template<> struct ConwayPolynomial<103, 1> { using ZPZ = aerobus::zpz<103>; using type =
03742
                          POLYV<ZPZV<1>, ZPZV<98»; }; // NOLINT
                                           template<> struct ConwayPolynomial<103, 2> { using ZPZ = aerobus::zpz<103>; using type =
03743
                          POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; }; // NOLINT
                                            template<> struct ConwayPolynomial<103, 3> { using ZPZ = aerobus::zpz<103>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<98»; }; // NOLINT
03745
                                          template<> struct ConwayPolynomial<103, 4> { using ZPZ = aerobus::zpz<103>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<88>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
03746
                         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 =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98»; }; // NOLINT
03749
                                          template<> struct ConwayPolynomial<103, 8> { using ZPZ = aerobus::zpz<103>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<70>, ZPZV<71>, ZPZV<71>, ZPZV<49>, ZPZV<5»; }; //
                          NOLINT
                                            template<> struct ConwayPolynomial<103, 9> { 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<97>, ZPZV<91>, ZPZV<98»; };
                           // NOLINT
                          template<> struct ConwayPolynomial<103, 10> { using ZPZ = aerobus::zpz<103>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<101>, ZPZV<86>, ZPZV<101>, ZPZV<94>, ZPZV<11>,
03751
                           ZPZV<5»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<103, 11> { using ZPZ = aerobus::zpz<103>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<5>, ZPZV<98»; }; // NOLINT</pre>
                          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<20>, ZPZV<81>,
03753
                           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
                           ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98»; };</pre>
                                                                                                                                                                                                                // NOLINT
03755
                                          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<0>, ZPZV<0>, ZPZV<102>, ZPZV<8>, ZPZV<89»; }; // NOLINT template<> struct ConwayPolynomial<103, 19> { 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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<9»; };</pre>
03757
                                           template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
                          POLYV<ZPZV<1>, ZPZV<105»; }; // NOLINT
                                            template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
03758
                          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 template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
03760
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; }; // NOLINT
                                            template<> struct ConwayPolynomial<107, 5> { using ZPZ = aerobus::zpz<107>; using type =
03761
                           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 =
03762
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<52>, ZPZV<22>, ZPZV<79>, ZPZV<2»; }; // NOLINT
03763
                                             template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type =
                         POLYV-ZPZV-1>, ZPZV-(>, ZPZV-(>, ZPZV-(>), ZPZV-(>), ZPZV-(>), ZPZV-(>), ZPZV-(>), ZPZV-(15); // NOLINT template<> struct ConwayPolynomial<107, 8> { using ZPZ = aerobus::zpz<107>; using type =
03764
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<24>, ZPZV<95>, ZPZV<2»; };
                                         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<3>, ZPZV<66>, ZPZV<105»; };
                           // NOLINT
                                           template<> struct ConwayPolynomial<107, 10> { using ZPZ = aerobus::zpz<107>; using type =
03766
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, 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<ZPZV<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>, ZPZV<61>, ZPZV<42>, ZPZV<57>, ZPZV<2»; }; // NOLINT
03768
                                         template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          template<> struct ConwayPolynomial<107, 17> { using ZPZ = aerobus::zpz<107>; using type =
03770
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<105»; };</pre>
                                           template<> struct ConwayPolynomial<107, 19> { 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>, ZPZV<24>, ZPZV<105»; }; //</pre>
                           NOLINT
                                           template<> struct ConwayPolynomial<109, 1> { using ZPZ = aerobus::zpz<109>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<103»; };
                                       template<> struct ConwayPolynomial<109, 2> { using ZPZ = aerobus::zpz<109>; using type =
                     POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
03774
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
03775
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; }; // NOLINT
                                       template<> struct ConwayPolynomial<109, 5> { using ZPZ = aerobus::zpz<109>; using type =
03776
                      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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<102>, ZPZV<66>, ZPZV<6»; }; // NOLINT
template<> struct ConwayPolynomial<109, 7> { using ZPZ = aerobus::zpz<109>; using type =
03777
03778
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<143, ZPZV<103»; };
                                       template<> struct ConwayPolynomial<109, 8> { using ZPZ = aerobus::zpz<109>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<34>, ZPZV<86>, ZPZV<6»; };
                                      template<> struct ConwayPolynomial<109, 9> { using ZPZ = aerobus::zpz<109>; using type =
03780
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<93>, ZPZV<87>, ZPZV<103»; };
                                       template<> struct ConwayPolynomial<109, 10> { using ZPZ = aerobus::zpz<109>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<71>, ZPZV<55>, ZPZV<16>, ZPZV<75>, ZPZV<69>,
                       ZPZV<6»; }; // NOLINT</pre>
03782
                                      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
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<50>, ZPZV<55>, ZPZV<55>, ZPZV<55>, ZPZV<65>,
                        ZPZV<103>, ZPZV<28>, ZPZV<6»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<109, 13> { using ZPZ = aerobus::zpz<109>; using type
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        template<> struct ConwayPolynomial<109, 17> { using ZPZ = aerobus::zpz<109>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       template<> struct ConwayPolynomial<109, 19> { using ZPZ = aerobus::zpz<109>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03786
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<15>, ZPZV<103w; }; //</pre>
03787
                                       template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYV<ZPZV<1>, ZPZV<110»; }; // NOLINT
03788
                                     template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
                      POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
03789
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<110»; }; // NOLINT
                                        template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<62>, ZPZV<3»; }; // NOLINT
03791
                                     template<> struct ConwayPolynomial<113, 5> { using ZPZ = aerobus::zpz<113>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110»; }; // NOLINT
03792
                                      template<> struct ConwayPolynomial<113, 6> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<59>, ZPZV<30>, ZPZV<71>, ZPZV<3»; }; // NOLINT
                                      template<> struct ConwayPolynomial<113, 7> { using ZPZ = aerobus::zpz<113>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
03794
                                     template<> struct ConwayPolynomial<113, 8> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<38>, ZPZV<28>, ZPZV<28, ZPZV<3»; }; //
                       NOLINT
03795
                                      template<> struct ConwayPolynomial<113, 9> { using ZPZ = aerobus::zpz<113>; using type =
                       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<108>, ZPZV<57>, ZPZV<45>, ZPZV<83>, ZPZV<56>,
                       ZPZV<3»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<3>, ZPZV<110»; };</pre>
                                                                                                                 // NOLINT
                                    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<98>, ZPZV<56>, ZPZV<10>, ZPZV<27>, ZPZV<3»; }; // NOLINT
                                      template<> struct ConwayPolynomial<113, 13> { using ZPZ = aerobus::zpz<113>; using type =
03799
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110»; };</pre>
                                                                                                                                                                                           // NOLINT
                                       template<> struct ConwayPolynomial<113, 17> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                           ZPZV<0>,
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110»; }; // NOLINT
template<> struct ConwayPolynomial<113, 19> { using ZPZ = aerobus::zpz<113>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03801
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2</pre>
03802
                                       template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
                       POLYV<ZPZV<1>, ZPZV<124»; }; // NOLINT
                                      template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
03803
                       POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3»; }; // NOLINT
                                       template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
03804
                      POLYV<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<124»; }; // NOLINT template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type =
03805
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<97>, ZPZV<3»; }; // NOLINT 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
03806
```

```
template<> struct ConwayPolynomial<127, 6> { using ZPZ = aerobus::zpz<127>; using type =
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<64>, ZPZV<115>, ZPZV<82>, ZPZV<83*, }; // 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<0>, ZPZV<15>, ZPZV<124»; }; // NOLINT template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type =
 03809
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<104>, ZPZV<55>, ZPZV<8>, ZPZV<3»; }; //
                                                      template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type
 03810
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<119>, ZPZV<126>, ZPZV<124»;
                                }; // NOLINT
03811
                                                   template<> struct ConwayPolynomial<127, 10> { using ZPZ = aerobus::zpz<127>; using type =
                                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<64>, ZPZV<95>, ZPZV<60>, ZPZV<4>,
                                ZPZV<3»; }; // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<11>, ZPZV<124»; }; // NOLINT</pre>
                                                      template<> struct ConwayPolynomial<127, 12> { using ZPZ = aerobus::zpz<127>; using type =
                               POLYYCZPZVC1>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC1>, ZPZVC119>, ZPZVC25>, ZPZVC33>, ZPZVC97>, ZPZVC15>, ZPZVC99>, ZPZVC8>, ZPZVC8>, ZPZVC8>, ZPZVC8>, ZPZVC8>, ZPZVC8>, ZPZVC9>, ZPZVC9>, ZPZVC9>, ZPZVC9>, ZPZVC9>, ZPZVC9>, ZPZVC8>, ZPZVC8>,
                                                     template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                   template<> struct ConwayPolynomial<127, 17> { using ZPZ = aerobus::zpz<127>; using type =
03815
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>
                                                                                                                                                                                                                                                                                                                                                                                                                                                  // NOLINT
                                                      template<> struct ConwayPolynomial<127, 19> { 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<0>, ZPZV<30>, ZPZV<30>,
                                NOLINT
03817
                                                    template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
                               POLYV<ZPZV<1>, ZPZV<129»; }; // NOLINT
 03818
                                                      template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
                                POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2»; }; // NOLINT
 03819
                                                 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 =
 03820
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<131, 5> { using ZPZ = aerobus::zpz<131>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<129»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<131, 6> { using ZPZ = aerobus::zpz<131>; using type =
 03822
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<4>, ZPZV<22>, ZPZV<22>, ; }; // NOLINT
                                                    template<> struct ConwayPolynomial<131, 7> { using ZPZ = aerobus::zpz<131>; using type =
03823
                               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 =
03824
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<72>, ZPZV<116>, ZPZV<104>, ZPZV<2»; };
                                NOLINT
03825
                                                  template<> struct ConwayPolynomial<131, 9> { using ZPZ = aerobus::zpz<131>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<19>, ZPZV<129»; };
                                 // NOLINT
                                                    template<> struct ConwayPolynomial<131, 10> { using ZPZ = aerobus::zpz<131>; using type =
03826
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<9>, ZPZV<9>, ZPZV<426>, ZPZV<44>,
                                ZPZV<2»; }; // NOLINT</pre>
03827
                                                  template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type :
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03828
                                                     template<> struct ConwayPolynomial<131, 12> { using ZPZ = aerobus::zpz<131>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<122>, ZPZV<40>, ZPZV<83>, ZPZV<125>,
                                ZPZV<28>, ZPZV<103>, ZPZV<2»; }; // NOLINT</pre>
                                                   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»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                template<> struct ConwayPolynomial<131, 19> { using ZPZ = aerobus::zpz<131>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 \texttt{ZPZV} < \texttt{0} >, \ \texttt{Z
                                NOLINT
                                                    template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
                               POLYV<ZPZV<1>, ZPZV<134»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<137, 2> { using ZPZ = aerobus::zpz<137>; using type =
                               POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type =
 03834
                               POLYY<ZPZY<1>, ZPZY<0>, ZPZY<6>, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
03835
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<95>, ZPZV<3»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
 03836
                               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 =
 03837
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<116>, ZPZV<102, ZPZV<3>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<137, 7> { using ZPZ = aerobus::zpz<137>; using type = DNIV(ZPZV)
 03838
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<134»; };
                                                    template<> struct ConwayPolynomial<137, 8> { using ZPZ = aerobus::zpz<137>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105>, ZPZV<21>, ZPZV<34>, ZPZV<3*, };
                               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<80>, ZPZV<122>, ZPZV<134»;</pre>
```

```
}; // NOLINT
template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<20>, ZPZV<67>, ZPZV<67>, ZPZV<93>, ZPZV<119>,
                  ZPZV<3»; }; // NOLINT</pre>
                 template<> struct ConwayPolynomial<137, 11> { using ZPZ = aerobus::zpz<137>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<134_{\odot}; // NOLINT
03842
                              template<> struct ConwayPolynomial<137, 12> { using ZPZ = aerobus::zpz<137>; using type
                  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 =
03844
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                 ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<134»; }; // NOLINT
   template<> struct ConwayPolynomial<137, 17> { using ZPZ = aerobus::zpz<137>; using type
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<4>, ZPZV<134»; }; // NOLINT
    template<> struct ConwayPolynomial<137, 19> { using ZPZ = aerobus::zpz<137>; 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<18>, ZPZV<18</pre>
                              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 =
03848
                POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2»; }; // NOLINT
03849
                             template<> struct ConwayPolynomial<139, 3> { using ZPZ = aerobus::zpz<139>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<137»; }; // NOLINT template<> struct ConwayPolynomial<139, 4> { using ZPZ = aerobus::zpz<139>; using type =
03850
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2»; }; // NOLINT
03851
                             template<> struct ConwayPolynomial<139, 5> { using ZPZ = aerobus::zpz<139>; using type =
                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 =
03852
                POLYV<ZPZV<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 =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<137»; }; // NOLINT
03854
                            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»; };
                 NOLINT
                             template<> struct ConwayPolynomial<139, 9> { using ZPZ = aerobus::zpz<139>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<70>, ZPZV<87>, ZPZV<137»; };
03856
                            template<> struct ConwayPolynomial<139, 10> { using ZPZ = aerobus::zpz<139>; using type =
                 POLYV<ZPZV<1>, 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 =
03857
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<137»; }; // NOLINT
                                                                                       // NOLINT
03858
                           template<> struct ConwayPolynomial<139, 12> { using ZPZ = aerobus::zpz<139>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<120>, ZPZV<75>, ZPZV<41>, ZPZV<41>, ZPZV<77>, ZPZV<106>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
                             template<> struct ConwayPolynomial<139, 13> { using ZPZ = aerobus::zpz<139>; using type =
03859
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<137»; }; // NOLINT</pre>
03860
                           template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
03861
                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<137»; }; //</pre>
03862
                             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 =
03863
                POLYV<ZPZV<1>, ZPZV<145>, ZPZV<2»; }; // NOLINT
                             template<> struct ConwayPolynomial<149, 3> { using ZPZ = aerobus::zpz<149>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
03865
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<149, 5> { using ZPZ = aerobus::zpz<149>; using type =
03866
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<147»; }; // NOLINT
                              template<> struct ConwayPolynomial<149, 6> { using ZPZ = aerobus::zpz<149>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<105>, ZPZV<33>, ZPZV<55>, ZPZV<2»; };
03868
                           template<> struct ConwayPolynomial<149, 7> { using ZPZ = aerobus::zpz<149>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 8> { using ZPZ = aerobus::zpz<149>; using type =
03869
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<140>, ZPZV<25>, ZPZV<123>, ZPZV<2»; }; //
03870
                           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<0>, ZPZV<146>, ZPZV<20>, ZPZV<147»;
                  }; // NOLINT
03871
                            template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<42>, ZPZV<148>, ZPZV<143>, ZPZV<51>,
                 ZPZV<2»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<149, 11> { using ZPZ = aerobus::zpz<149>; using type
                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                 \text{ZPZV}<33>, \text{ZPZV}<147»; }; // NOLINT
                 template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<91>, ZPZV<52>, ZPZV<52>,
03873
```

```
ZPZV<104>, ZPZV<110>, ZPZV<2»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<149, 13> { using ZPZ = aerobus::zpz<149>; using type =
                               POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<147»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03875
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<147»; }; // NOLINT</pre>
03876
                                                  template<> struct ConwayPolynomial<149, 19> { using ZPZ = aerobus::zpz<149>; 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>, ZPZV<5>, ZPZV<147»; }; //
                               NOLINT
                                                  template<> struct ConwayPolynomial<151, 1> { using ZPZ = aerobus::zpz<151>; using type =
03877
                              POLYV<ZPZV<1>, ZPZV<145»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<151, 2> { using ZPZ = aerobus::zpz<151>; using type =
                               POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6»; }; // NOLINT
 03879
                                                  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 =
03880
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<151, 5> { using ZPZ = aerobus::zpz<151>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<145»; }; // NOLINT
 03882
                                                 template<> struct ConwayPolynomial<151, 6> { using ZPZ = aerobus::zpz<151>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<15>, ZPZV<15>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<151, 7> { using ZPZ = aerobus::zpz<151>; using type =
03883
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9+, ZPZV<145»; };
                                                   template<> struct ConwayPolynomial<151, 8> { using ZPZ = aerobus::zpz<151>, using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<140>, ZPZV<122>, ZPZV<43>, ZPZV<6»; };
                               NOLINT
03885
                                                  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
03886
                                                     template<> struct ConwayPolynomial<151, 10> { using ZPZ = aerobus::zpz<151>; using type
                               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<21>, ZPZV<104>, ZPZV<49>, ZPZV<49>, ZPZV<20, ZPZV<14>,
                               ZPZV<6»; }; // NOLINT</pre>
                               \label{eq:convayPolynomial} $$ \text{template} <> \text{struct ConwayPolynomial} <151, 11> { using ZPZ = aerobus::zpz<151>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
03887
                               ZPZV<1>, ZPZV<145»; };</pre>
                                                                                                                                                     // NOLINT
                                                    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<121>, ZPZV<101>, ZPZV<6>, ZPZV<77>,
                                ZPZV<107>, ZPZV<147>, ZPZV<6»; }; // NOLINT</pre>
03889
                                                 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 =
03890
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<145»; }; // NOLINT</pre>
03891
                                                 template<> struct ConwayPolynomial<151, 19> { using ZPZ = aerobus::zpz<151>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                   template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
                               POLYV<ZPZV<1>, ZPZV<152»; }; // NOLINT
 03893
                                                  template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
                               POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<157, 3> { using ZPZ = aerobus::zpz<157>; using type =
03894
                              POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<152»; }; // NOLINT template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
                              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
03897
                                                 template<> struct ConwayPolynomial<157, 6> { using ZPZ = aerobus::zpz<157>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<130>, ZPZV<144>, ZPZV<5»; ); // NOLINT template<> struct ConwayPolynomial<157, 7> { using ZPZ = aerobus::zpz<157>; using type
 03898
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; }; //
03899
                                                 template<> struct ConwayPolynomial<157, 8> { using ZPZ = aerobus::zpz<157>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<97>, ZPZV<40>, ZPZV<153>, ZPZV<5»; };
                               NOLINT
03900
                                                  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<2>, ZPZV<114>, ZPZV<52>, ZPZV<152»;
                               }; // NOLINT
                                                  template<> struct ConwayPolynomial<157, 10> { using ZPZ = aerobus::zpz<157>; using type =
03901
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, 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 =
03902
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                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<0>, ZPZV<1>, ZPZV<110>, ZPZV<12>, ZPZV<12>, ZPZV<137>, ZPZV<137>, ZPZV<137>, ZPZV<10>, ZPZV
                               ZPZV<152>, ZPZV<57>, ZPZV<5»; }; // NOLINT</pre>
03904
                                                 template<> struct ConwayPolynomial<157, 13> { using ZPZ = aerobus::zpz<157>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<156>, ZPZV<9>, ZPZV<152»; }; // NOLINT
  template<> struct ConwayPolynomial<157, 17> { using ZPZ = aerobus::zpz<157>; using type
                               POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<12>, ZPZV<152»; }; // NOLINT
    template<> struct ConwayPolynomial<157, 19> { using ZPZ = aerobus::zpz<157>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
 03906
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<152»; }; //</pre>
03907
                                   template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
                      POLYV<ZPZV<1>, ZPZV<161»; }; // NOLINT
03908
                                     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 =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<161»; }; // NOLINT
                                   template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
03910
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2»; }; // NOLINT
                                     template<> struct ConwayPolynomial<163, 5> { using ZPZ = aerobus::zpz<163>; using type =
03911
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<161»; }; // NOLINT
03912
                                     template<> struct ConwayPolynomial<163, 6> { using ZPZ = aerobus::zpz<163>; using type =
                      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<25>, ZPZV<156>, ZPZV<2»; }; // NOLINT
03913
                                   template<> struct ConwayPolynomial<163, 7> { using ZPZ = aerobus::zpz<163>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<161»; }; // NOLINT template<> struct ConwayPolynomial<163, 8> { using ZPZ = aerobus::zpz<163>; using type =
03914
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<132>, ZPZV<83>, ZPZV<6>, ZPZV<2»; };
                                     template<> struct ConwayPolynomial<163, 9> { using ZPZ = aerobus::zpz<163>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>>, ZPZV<127>, ZPZV<161»;
                      }; // NOLINT
03916
                                     template<> struct ConwayPolynomial<163, 10> { using ZPZ = aerobus::zpz<163>; using type =
                      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
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<11>, ZPZV<161»; }; // NOLINT</pre>
03918
                                      template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<112>, ZPZV<31>, ZPZV<38>, ZPZV<103>, ZPZV<10>, ZPZV<69>, ZPZV<2»; }; // NOLINT
03919
                                      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<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
03920
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<161»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<163, 19> { using ZPZ = aerobus::zpz<163>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<8>, ZPZV<161»; };</pre>
                      NOLINT
03922
                                     template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
                      POLYV<ZPZV<1>, ZPZV<162»; }; // NOLINT
03923
                                     template<> struct ConwayPolynomial<167, 2> { using ZPZ = aerobus::zpz<167>; using type =
                      POLYV<ZPZV<1>, ZPZV<166>, ZPZV<5»; }; // NOLINT
03924
                                     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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 5> { using ZPZ = aerobus::zpz<167>; using type =
03925
03926
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<162»; }; // NOLINT
                                      template<> struct ConwayPolynomial<167, 6> { using ZPZ = aerobus::zpz<167>; using type =
                       \texttt{POLYV} < \texttt{ZPZV} < 1 >, \ \texttt{ZPZV} < 0 >, \ \texttt{ZPZV} < 2 >, \ \texttt{ZPZV} < 75 >, \ \texttt{ZPZV} < 38 >, \ \texttt{ZPZV} < 2 >, \ \texttt{ZPZV} < 5 >; \ \}; \ \ // \ \ \texttt{NOLINT} 
03928
                                    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<10>, ZPZV<162»; }; // NOLINT
                                     template<> struct ConwayPolynomial<167, 8> { using ZPZ = aerobus::zpz<167>; using type =
03929
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<149>, ZPZV<56>, ZPZV<113>, ZPZV<55»; };
                      template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<165>, ZPZV<162»;
                      }; // NOLINT
                                      template<> struct ConwayPolynomial<167, 10> { using ZPZ = aerobus::zpz<167>; using type =
03931
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68>, ZPZV<68>, ZPZV<109>, ZPZV<143>,
                                                                                                            // NOLINT
                      ZPZV<148>, ZPZV<5»; };</pre>
                                   template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<24>, ZPZV<162»; }; // NOLINT
                                    template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type =
03933
                      POLYV<PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<142>, ZPZV<142>, ZPZV<142>, ZPZV<131>,
                      ZPZV<140>, ZPZV<41>, ZPZV<57>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 13> { using ZPZ = aerobus::zpz<167>; using type :
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<10>, ZPZV<16>, ZPZV<16>; // NOLINT
template<> struct ConwayPolynomial<167, 17> { using ZPZ = aerobus::zpz<167>; using type
03935
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<36*, };</pre>
                                   template<> struct ConwayPolynomial<167, 19> { using ZPZ = aerobus::zpz<167>; using type
03936
                      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<14>, ZPZV<162»; }; //</pre>
                      NOLINT
03937
                                     template<> struct ConwayPolynomial<173, 1> { using ZPZ = aerobus::zpz<173>; using type =
                     POLYV<ZPZV<1>, ZPZV<171»; };
                                                                                                                                    // NOLINT
                                      template<> struct ConwayPolynomial<173, 2> { using ZPZ = aerobus::zpz<173>; using type =
                     POLYV<ZPZV<1>, ZPZV<169>, ZPZV<2»; }; // NOLINT
03939
                                   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 =
03940
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<2»; };
                                      template<> struct ConwayPolynomial<173, 5> { 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, 6> { using ZPZ = aerobus::zpz<173>; using type =
03942
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<134>, ZPZV<107>, ZPZV<2»; }; // NOLINT
03943
                                    template<> struct ConwayPolynomial<173, 7> { using ZPZ = aerobus::zpz<173>; using type
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<171»; }; //
03944
                                     template<> struct ConwayPolynomial<173, 8> { using ZPZ = aerobus::zpz<173>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<125>, ZPZV<158>, ZPZV<27>, ZPZV<27>, ZPZV<2»; }; //
03945
                                   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<56>, ZPZV<56>, ZPZV<104>, ZPZV<171»;
                      }; // NOLINT
    template<> struct ConwayPolynomial<173, 10> { using ZPZ = aerobus::zpz<173>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<156>, ZPZV<164>, ZPZV<48>, ZPŽV<106>,
                      ZPZV<58>, ZPZV<2»; };</pre>
                                                                                                           // NOLINT
                                      template<> struct ConwayPolynomial<173, 11> { using ZPZ = aerobus::zpz<173>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<171»; }; // NOLINT
                                      template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type
                      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
03949
                                    template<> struct ConwayPolynomial<173, 13> { using ZPZ = aerobus::zpz<173>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171»; };
                                                                                                                                                                                  // NOLINT
                                      template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 7>, \ \texttt{ZPZV} < 71 \\ \texttt{n}; \ \ // \ \ \texttt{NOLINT} 
                                     template<> struct ConwayPolynomial<173, 19> { using ZPZ = aerobus::zpz<173>; using type =
03951
                      POLYV<ZPZV<1>, 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
                                     template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
03952
                      POLYV<ZPZV<1>, ZPZV<177»; }; // NOLINT
03953
                                     template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
                      POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<179, 3> { using ZPZ = aerobus::zpz<179>; using type =
03954
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<177»; }; // NOLINT
03955
                                     template<> struct ConwayPolynomial<179, 4> { using ZPZ = aerobus::zpz<179>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<2w; }; // NOLINT template<> struct ConwayPolynomial<179, 5> { using ZPZ = aerobus::zpz<179>; using type =
03956
                    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 =
03957
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<91>, ZPZV<55>, ZPZV<109>, ZPZV<2»; }; // NOLINT
                                     template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<177»; };
03959
                                  template<> struct ConwayPolynomial<179, 8> { using ZPZ = aerobus::zpz<179>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<163>, ZPZV<144>, ZPZV<73>, ZPZV<2»; }; //
                      NOLINT
                                    template<> struct ConwayPolynomial<179, 9> { using ZPZ = aerobus::zpz<179>; using type =
03960
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<54>, ZPZV<54
03961
                                    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
03962
                                     template<> struct ConwayPolynomial<179, 11> { using ZPZ = aerobus::zpz<179>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                               // NOLINT
                      ZPZV<28>, ZPZV<177»; };</pre>
                      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<43>, ZPZV<6>, ZPZV<8>,
                      ZPZV<177>, ZPZV<1>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<179, 13> { using ZPZ = aerobus::zpz<179>; using type =
03964
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<177»; };</pre>
                                                                                                                                                                                  // NOLINT
                                  template<> struct ConwayPolynomial<179, 17> { 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<4>, ZPZV<477»; }; // NOLINT
template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type =
03966
                      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<11>, ZPZV<177»; }; //</pre>
                                    template<> struct ConwayPolynomial<181, 1> { using ZPZ = aerobus::zpz<181>; using type =
                      POLYV<ZPZV<1>, ZPZV<179»; }; // NOLINT
                                     template<> struct ConwayPolynomial<181, 2> { using ZPZ = aerobus::zpz<181>; using type =
03968
                      POLYV<ZPZV<1>, ZPZV<177>, ZPZV<2»; }; // NOLINT
                                      template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179»; }; // NOLINT template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
03970
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<105>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
03971
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<179»; }; // NOLINT
03972
                                     template<> struct ConwayPolynomial<181, 6> { using ZPZ = aerobus::zpz<181>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<163>, ZPZV<169>, ZPZV<2; }; // NOLINT template<> struct ConwayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type =
03973
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<179»; }; // NOLINT 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»; }; //
03974
```

```
NOLINT
03975
                                     template<> struct ConwayPolynomial<181, 9> { using ZPZ = aerobus::zpz<181>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<107>, ZPZV<168>, ZPZV<179»;
                      }; // NOLINT
                                       template<> struct ConwayPolynomial<181, 10> { using ZPZ = aerobus::zpz<181>; using type =
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>4>, ZPZV<104>, ZPZV<94>, ZPZV<98>, ZPZV<88>,
                      ZPZV<2»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<181, 11> { using ZPZ = aerobus::zpz<181>; using type :
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<24>, ZPZV<179»; }; // NOLINT</pre>
03978
                                    template<> struct ConwayPolynomial<181, 12> { using ZPZ = aerobus::zpz<181>; using type =
                      POLYV<2PZV<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
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      /<0>, ZPZV<0>, ZPZV<8>, ZPZV<179»; }; // NOLINT
template<> struct ConwayPolynomial<181, 17> { using ZPZ = aerobus::zpz<181>; using type =
                       ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<179»; };</pre>
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<179»; };</pre>
                                                                                                                                                                                                                                                                                                                                // NOLINT
                                      template<> struct ConwayPolynomial<181, 19> { using ZPZ = aerobus::2pz<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<0>, ZPZV<0>, ZPZV<36>, ZPZV<179»; }; //</pre>
                       NOLINT
                                     template<> struct ConwayPolynomial<191, 1> { using ZPZ = aerobus::zpz<191>; using type =
03982
                      POLYV<ZPZV<1>, ZPZV<172»; }; // NOLINT
                                     template<> struct ConwayPolynomial<191, 2> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19»; }; // NOLINT
                                     template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
03984
                    POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172»; ); // NOLINT
template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; }; // NOLINT
template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
03985
03986
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<172»; }; // NOLINT
03987
                                    template<> struct ConwayPolynomial<191, 6> { using ZPZ = aerobus::zpz<191>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<110>, ZPZV<10>, ZPZV<10>, ZPZV<19»; }; // NOLINT template<> struct ConwayPolynomial<191, 7> { using ZPZ = aerobus::zpz<191>; using type =
03988
                      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1+>, 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»; }; //
03990
                                    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<62>, ZPZV<62>, ZPZV<62>, ZPZV<124>, ZPZV<172»;
                       }; // NOLINT
03991
                                       template<> struct ConwayPolynomial<191, 10> { using ZPZ = aerobus::zpz<191>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<47>, ZPZV<47>, ZPZV<173>, ZPZV<74>,
                       ZPZV<156>, ZPZV<19»; }; // NOLINT</pre>
03992
                                    template<> struct ConwayPolynomial<191, 11> { using ZPZ = aerobus::zpz<191>; using type =
                       \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ 
                       ZPZV<6>, ZPZV<172»: }: // NOLINT</pre>
                                      template<> struct ConwayPolynomial<191, 12> { using ZPZ = aerobus::zpz<191>; using type
03993
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<168>, ZPZV<25>, ZPZV<25>, ZPZV<49>, ZPZV<90>,
                       ZPZV<7>, ZPZV<151>, ZPZV<19»; }; // NOLINT</pre>
03994
                                    template<> struct ConwayPolynomial<191, 13> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<172»; }; // NOLINT template<> struct ConwayPolynomial<191, 17> { using ZPZ = aerobus::zpz<191>; using type =
03995
                       POLYV<ZPZV<1>, ZPZV<0>, 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<190>, ZPZV<2>, ZPZV<172»; }; //
                       NOLINT
03997
                                      template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
                      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
03999
                                      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 =
04000
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<5»; }; // NOLINT
                                     template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
04001
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<188»; }; // NOLINT
                                      \texttt{template<> struct ConwayPolynomial<193, 6> \{ using \ ZPZ = aerobus::zpz<193>; using \ type = aerobus::zp
04002
                      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
04003
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<188»; }; // NOLINT
                                    template<> struct ConwayPolynomial<193, 8> { using ZPZ = aerobus::zpz<193>; using type =
04004
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<145>, ZPZV<34>, ZPZV<154>, ZPZV<5»; }; //
                      NOLINT
04005
                                    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<168>, ZPZV<27>, ZPZV<188»;
                      }; // NOLINT
template<> struct ConwayPolynomial<193, 10> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<51>, ZPZV<77>, ZPZV<0>, ZPZV<89>,
                      ZPZV<5»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type =
04007
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
```

```
ZPZV<1>, ZPZV<188»; };</pre>
                                           template<> struct ConwayPolynomial<193, 12> { using ZPZ = aerobus::zpz<193>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<155>, ZPZV<52>, ZPZV<135>, ZPZV<135>, ZPZV<155>, ZPZV<135>, ZPZV<
                          ZPZV<90>, ZPZV<46>, ZPZV<28>, ZPZV<5»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<193, 13> { using ZPZ = aerobus::zpz<193>; using type =
04009
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                            // NOLINT
                                           template<> struct ConwayPolynomial<193,
                                                                                                                                                                                                                    17> { using ZPZ = aerobus::zpz<193>; using type
04010
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<188»; }; // NOLINT
    template<> struct ConwayPolynomial<193, 19> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04011
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5</pre>
04012
                                         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 =
04013
                         POLYVCZPZV<1>, ZPZV<192>, ZPZV<2»; }; // NOLINT 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 =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<124>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<197, 5> { using ZPZ = aerobus::zpz<197>; using type =
04016
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195»; }; // NOLINT
04017
                                           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
 04018
                                         template<> struct ConwayPolynomial<197, 7> { using ZPZ = aerobus::zpz<197>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<195»; }; // NOLINT template<> struct ConwayPolynomial<197, 8> { using ZPZ = aerobus::zpz<197>; using type =
 04019
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<176>, ZPZV<96>, ZPZV<29>, ZPZV<2»; };
                          NOLINT
04020
                                           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<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<73>, ZPZV<42>,
 04021
                          ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<197, 11> { using ZPZ = aerobus::zpz<197>; using type
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<14>, ZPZV<195»; }; // NOLINT</pre>
04023
                                         template<> struct ConwayPolynomial<197, 12> { using ZPZ = aerobus::zpz<197>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<168>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>,
                          ZPZV<90>, ZPZV<163>, ZPZV<2»; }; // NOLINT</pre>
04024
                                          template<> struct ConwayPolynomial<197, 13> { using ZPZ = aerobus::zpz<197>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<195»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<197, 17> { using ZPZ = aerobus::zpz<197>; using type =
04025
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04026
                           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<6>, ZPZV<6</pre>, <pre
04027
                                           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 =
04028
                          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 =
 04030
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
04031
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
                                           template<> struct ConwayPolynomial<199, 6> { using ZPZ = aerobus::zpz<199>; using type =
                         POLYV<ZPZV<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 =
 04033
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT template<> struct ConwayPolynomial<199, 8> { using ZPZ = aerobus::zpz<199>; using type =
04034
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<160>, ZPZV<23>, ZPZV<159>, ZPZV<3»; };
                          NOLINT
                                           template<> struct ConwayPolynomial<199, 9> { using ZPZ = aerobus::zpz<199>; using type =
04035
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<141>, ZPZV<196»;
                          }; // NOLINT
                                           template<> struct ConwayPolynomial<199, 10> { using ZPZ = aerobus::zpz<199>; using type =
04036
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<158>, ZPZV<31>, ZPZV<54>, ZPZV<9>,
                          ZPZV<3»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<199, 11> { using ZPZ = aerobus::zpz<199>; using type :
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; }; // NOLINT
                                        template<> struct ConwayPolynomial<199, 12> { using ZPZ = aerobus::zpz<199>; using type =
04038
                          POLYV<ZPZV<1>, ZPZV<0>, 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
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<196»; }; // NOLINT
   template<> struct ConwayPolynomial<199, 17> { using ZPZ = aerobus::zpz<199>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04040
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<136»; };</pre>
                      template<> struct ConwayPolynomial<199, 19> { using ZPZ = aerobus::zpz<199>; 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<19>, ZPZV<196»; }; //</pre>
                       NOLINT
04042
                                      template<> struct ConwavPolynomial<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 =
04043
                      POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; }; // NOLINT
04044
                                    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 =
04045
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2»; };
                                                                                                                                                                                                                                             // NOLINT
                                      template<> struct ConwayPolynomial<211, 5> { using ZPZ = aerobus::zpz<211>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<209»; }; // NOLINT
04047
                                      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
                                      template<> struct ConwayPolynomial<211, 7> { using ZPZ = aerobus::zpz<211>; using type
04048
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<209»; }; // NOLINT
                                     template<> struct ConwayPolynomial<211, 8> { using ZPZ = aerobus::zpz<211>; using type =
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<200>, ZPZV<87>, ZPZV<29>, ZPZV<29; };
                       NOLINT
04050
                                    template<> struct ConwayPolynomial<211, 9> { using ZPZ = aerobus::zpz<211>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<19>, ZPZV<19>, ZPZV<13>, ZPZV<26>, ZPZV<20>;
                       }; // NOLINT
                                       template<> struct ConwayPolynomial<211, 10> { using ZPZ = aerobus::zpz<211>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<30>, ZPZV<61>, ZPZV<148>, ZPZV<148>, ZPZV<125>,
                       ZPZV<2»; }; // NOLINT</pre>
04052
                                      template<> struct ConwayPolynomial<211, 11> { using ZPZ = aerobus::zpz<211>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04053
                                      template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<50>, ZPZV<126>, ZPZV<184>, ZPZV<84>, ZPZV<27>, ZPZV<22>; }; // NOLINT
                      \label{eq:convayPolynomial} $$ \text{template} > \text{struct ConwayPolynomial} < 11, 13> { using ZPZ = aerobus::zpz<211>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZP
04054
                      ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<209»; }; // NOLINT
template<> struct ConwayPolynomial<211, 17> { using ZPZ = aerobus::zpz<211>; using type
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<1209»; }; // NOLINT</pre>
04056
                                    template<> struct ConwayPolynomial<211, 19> { using ZPZ = aerobus::zpz<211>; using type =
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      template<> struct ConwayPolynomial<223, 1> { using ZPZ = aerobus::zpz<223>; using type =
                      POLYV<ZPZV<1>, ZPZV<220»; }; // NOLINT
04058
                                   template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
                     POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; }; // NOLINT
04059
                                      template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
                     POLYVCZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
04060
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<3»; }; // NOLINT
04061
                                    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 template<> struct ConwayPolynomial<223, 6> { using ZPZ = aerobus::zpz<223>; using type =
04062
                     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
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<220»; };
                                    template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<3»; }; //
                      NOLINT
                                    template<> struct ConwayPolynomial<223, 9> { using ZPZ = aerobus::zpz<223>; using type =
04065
                       POLYV<ZPZV<1>, ZPZV<0>, 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 =
04066
                       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>
                                    template<> struct ConwayPolynomial<223, 11> { using ZPZ = aerobus::zpz<223>; using type =
04067
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<8>, ZPZV<220»; };</pre>
                                                                                                                // NOLINT
                                      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>,
                       template<> struct ConwayPolynomial<223, 13> { using ZPZ = aerobus::zpz<223>; using type :
04069
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       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<1>, ZPZV<220»; }; // NOLINT
    template<> struct ConwayPolynomial<223, 19> { using ZPZ = aerobus::zpz<223>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04071
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7</pre>; ;;
04072
                                    template<> struct ConwayPolynomial<227, 1> { using ZPZ = aerobus::zpz<227>; using type =
                     POLYV<ZPZV<1>, ZPZV<225»; }; // NOLINT template<> struct ConwayPolynomial<227, 2> { using ZPZ = aerobus::zpz<227>; using type =
04073
                       POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
                    POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<225»; }; // NOLINT template<> struct ConwayPolynomial<227, 4> { using ZPZ = aerobus::zpz<227>; using type =
04075
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<13>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<227, 5> { using ZPZ = aerobus::zpz<227>; using type =
 04076
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<25»; }; // NOLINT
                                   template<> struct ConwayPolynomial<227, 6> { using ZPZ = aerobus::zpz<227>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<24>, ZPZV<135>, ZPZV<2»; }; // NOLINI
 04078
                                  template<> struct ConwayPolynomial<227, 7> { using ZPZ = aerobus::zpz<227>; using type =
                    POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<225»; }; // NoLT template<> struct ConwayPolynomial<227, 8> { using ZPZ = aerobus::zpz<227>; using type =
04079
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<151>, ZPZV<176>, ZPZV<106>, ZPZV<2»; }; //
                                    template<> struct ConwayPolynomial<227, 9> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<24>, ZPZV<183>, ZPZV<225»;
                      }; // NOLINT
                                   template<> struct ConwayPolynomial<227, 10> { using ZPZ = aerobus::zpz<227>; using type =
04081
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>, ZPZV<12>, ZPZV<12>, ZPZV<7>, ZPZV<7>,
                      ZPZV<2»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<2>, ZPZV<225»; }; // NOLINT
  template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; using type =
04083
                     POLYV<ZPZV<1>, ZPZV<0>, 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 = aerobus:
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<225»; };  // NOLINT</pre>
04085
                                    template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
04086
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<225»; }; //</pre>
                      NOLINT
                                   template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
04087
                     POLYV<ZPZV<1>, ZPZV<223»; }; // NOLINT
                                   template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
                     POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»; }; // NOLINT
                                  template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; }; // NOLINT template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
04090
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<229, 5> { using ZPZ = aerobus::zpz<229>; using type =
 04091
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223»; }; // NOLINT
 04092
                                   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*; }; // NOLINT
04093
                                  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<223»; }; // NOLINT
                                  template<> struct ConwayPolynomial<229, 8> { using ZPZ = aerobus::zpz<229>; using type =
04094
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<193>, ZPZV<62>, ZPZV<205>, ZPZV<6»; }; //
04095
                                  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>,
04096
                      ZPZV<98>, ZPZV<6»; };
                                                                                                      // NOLINT
                                  template<> struct ConwayPolynomial<229, 11> { using ZPZ = aerobus::zpz<229>; 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<2>, ZPZV<223»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<229, 12> { using ZPZ = aerobus::zpz<229>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<131>, ZPZV<140>, ZPZV<25>, ZPZV<6>, ZPZV<172>, ZPZV<9>, ZPZV<6»; }; // NOLINT
                                 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>
                                  template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type =
04100
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<23»; };</pre>
                                                                                                                                                                                                                                                                                                        // NOLINT
                                   template<> struct ConwayPolynomial<229, 19> { using ZPZ = aerobus::zpz<229>; 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<223»; }; //</pre>
                      NOLINT
04102
                                   template<> struct ConwayPolynomial<233, 1> { using ZPZ = aerobus::zpz<233>; using type =
                      POLYV<ZPZV<1>, ZPZV<230»; }; // NOLINT
 04103
                                  template<> struct ConwayPolynomial<233, 2> { using ZPZ = aerobus::zpz<233>; using type =
                     POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
 04104
                                  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 =
 04105
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<3»; };
                                                                                                                                                                                                                            // NOLINT
                                   template<> struct ConwayPolynomial<233, 5> { using ZPZ = aerobus::zpz<233>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<230»; }; // NOLINT
 04107
                                 template<> struct ConwayPolynomial<233, 6> { using ZPZ = aerobus::zpz<233>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<122>, ZPZV<215>, ZPZV<32>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<233, 7> { using ZPZ = aerobus::zpz<233>; using type =
 04108
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<230»; };
                          template<> struct ConwayPolynomial<233, 8> { using ZPZ = aerobus::zpz<233>; using type
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<202>, ZPZV<135>, ZPZV<181>, ZPZV<3»; }; //
                NOLINT
04110
                          template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<56>, ZPZV<146>, ZPZV<230»;
                         template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type
04111
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<28>, ZPZV<71>, ZPZV<102>, ZPZV<3>, ZPZV<48>,
                ZPZV<3»; }; // NOLINT</pre>
04112
                         template<> struct ConwayPolynomial<233, 11> { using ZPZ = aerobus::zpz<233>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                ZPZV<5>, ZPZV<230»; };</pre>
                                                                               // NOLINT
                          template<> struct ConwayPolynomial<233, 12> { using ZPZ = aerobus::zpz<233>; using type
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, 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 =
                POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                                                                                                                                   17> { using ZPZ = aerobus::zpz<233>; using type
                          template<> struct ConwayPolynomial<233,
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                                                                                                                                                                                                             // NOLINT
                template<> struct ConwayPolynomial<233, 19> { using ZPZ = aerobus::zpz<233>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2
04116
                          template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
04117
                POLYV<ZPZV<1>, ZPZV<232»; }; // NOLINT
04118
                          template<> struct ConwayPolynomial<239, 2> { using ZPZ = aerobus::zpz<239>; using type =
                POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7»; }; // NOLINT
                         template<> struct ConwayPolynomial<239, 3> { using ZPZ = aerobus::zpz<239>; using type =
04119
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<32»; }; // NOLINT template<> struct ConwayPolynomial<239, 4> { using ZPZ = aerobus::zpz<239>; using type =
04120
                 \verb"POLYV<ZPZV<1>, \ \verb"ZPZV<0>, \ \verb"ZPZV<11>, \ \verb"ZPZV<132>, \ \verb"ZPZV<7"; \ \verb"}; \ \ // \ \verb"NOLINT" 
04121
                          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
                          template<> struct ConwayPolynomial<239, 6> { using ZPZ = aerobus::zpz<239>; using type =
04122
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<237>, ZPZV<60>, ZPZV<200>, ZPZV<7»; }; // NOLINT
04123
                           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<0>, ZPZV<17>, ZPZV<232»; }; //
04124
                         template<> struct ConwayPolynomial<239, 8> { using ZPZ = aerobus::zpz<239>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<202>, ZPZV<54>, ZPZV<54>, ZPZV<7»; }; //
                NOLINT
04125
                         template<> struct ConwayPolynomial<239, 9> { using ZPZ = aerobus::zpz<239>; using type
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<88>, ZPZV<3232»; };
                 // NOLINT
04126
                         template<> struct ConwayPolynomial<239, 10> { using ZPZ = aerobus::zpz<239>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, 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 =
04127
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                ZPZV<8>, ZPZV<232»; }; // NOLINT</pre>
04128
                         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<182>, ZPZV<101>, ZPZV<81>, ZPZV<216>, ZPZV<7»; }; // NOLINT
                          template<> struct ConwayPolynomial<239, 13> { using ZPZ = aerobus::zpz<239>; using type :
04129
                POLYVCDZVV(1>, ZPZV<0>, ZPZV<0
                ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; };</pre>
                                                                                                                              // NOLINT
                          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>, ZPZV<0>, ZPZV<0>,
                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<232»; }; // NOLINT
template<> struct ConwayPolynomial<239, 19> { 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>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<232»; }; //</pre>
04132
                         template<> struct ConwayPolynomial<241, 1> { using ZPZ = aerobus::zpz<241>; using type =
                POLYV<ZPZV<1>, ZPZV<234»; }; // NOLINT
                         template<> struct ConwayPolynomial<241, 2> { using ZPZ = aerobus::zpz<241>; using type =
04133
               POLYV<ZPZV<1>, ZPZV<238>, ZPZV<7»; }; // NOLINT
                           template<> struct ConwayPolynomial<241, 3> { using ZPZ = aerobus::zpz<241>; using type =
04134
                POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZV<234»; }; // NOLINT template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
04135
               POLYV<2PZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<15>, ZPZV<15>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<241, 5> { using ZPZ = aerobus::zpz<241>; using type =
04136
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<234»; }; // NOLINT
                           template<> struct ConwayPolynomial<241, 6> { using ZPZ = aerobus::zpz<241>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<6>, ZPZV<5>, ZPZV<7»; }; // NOLINT
                         template<> struct ConwayPolynomial<241, 7> { using ZPZ = aerobus::zpz<241>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<241>; using type = template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
04139
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<113>, ZPZV<212>, ZPZV<153>, ZPZV<7»; }; //
                         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<236>, ZPZV<125>, ZPZV<234»;
                }; // NOLINT
                          template<> struct ConwayPolynomial<241, 10> { using ZPZ = aerobus::zpz<241>; using type =
04141
                POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<27>, ZPZV<145>, ZPZV<145>, ZPZV<36>, ZPZV<55>,
```

```
ZPZV<7»; };
                                            template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<3>, ZPZV<234»; }; // NOLINT</pre>
                           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</pre>
04143
                                              template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type
04144
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234»; }; // NOLINT
template<> struct ConwayPolynomial<241, 17> { using ZPZ = aerobus::zpz<241>; using type =
04145
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<234*; }; // NOLINT
template<> struct ConwayPolynomial<241, 19> { 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<11>, ZPZV<234»; }; //</pre>
                            NOLINT
                                             template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
04147
                           POLYV<ZPZV<1>, ZPZV<245»; }; // NOLINT
                                            template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
                           POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; }; // NOLINT
04149
                                            template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<200>, ZPZV<6»; }; // NOLINT
 04150
                                             template<> struct ConwayPolynomial<251, 5> { using ZPZ = aerobus::zpz<251>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT
 04152
                                            template<> struct ConwayPolynomial<251, 6> { using ZPZ = aerobus::zpz<251>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<247>, ZPZV<151>, ZPZV<179>, ZPZV<6»; }; // NOLINT
04153
                                           template<> struct ConwayPolynomial<251, 7> { using ZPZ = aerobus::zpz<251>; using type =
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<245»; }; // NOLINT
 04154
                                             template<> struct ConwayPolynomial<251, 8> { using ZPZ = aerobus::zpz<251>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<142>, ZPZV<215>, ZPZV<173>, ZPZV<6»; }; //
                          template<> struct ConwayPolynomial<251, 9> { using ZPZ = aerobus::zpz<251>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<106>, ZPZV<245»;</pre>
04155
                           }; // 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>,
                            ZPZV<149>, ZPZV<6»; };</pre>
                                                                                                                                  // NOLINT
04157
                                           template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type =
                            \texttt{POLYV} < \texttt{ZPZV} < \texttt{0} >, \ \texttt{ZPZV} < \texttt{
                           ZPZV<26>, ZPZV<245»; }; // NOLINT
  template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type :
04158
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<19>, ZPZV<53>, ZPZV<20>, ZPZV<20>, ZPZV<15>,
                            ZPZV<201>, ZPZV<232>, ZPZV<6»; };  // NOLINT</pre>
04159
                                          template<> struct ConwayPolynomial<251, 13> { using ZPZ = aerobus::zpz<251>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<245»; }; // NOLINT template<> struct ConwayPolynomial<251, 17> { using ZPZ = aerobus::zpz<251>; using type =
04160
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<245»; };</pre>
04161
                                          template<> struct ConwayPolynomial<251, 19> { using ZPZ = aerobus::zpz<251>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<445»; }; //</pre>
                            NOLINT
                                             template<> struct ConwayPolynomial<257, 1> { using ZPZ = aerobus::zpz<257>; using type =
                           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
                                           template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
04164
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<254»; }; // NOLINT template<> struct ConwayPolynomial<257, 4> { using ZPZ = aerobus::zpz<257>; using type =
 04165
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<18>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<257, 5> { using ZPZ = aerobus::zpz<257>; using type =
04166
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<254»; }; // NOLINT
 04167
                         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</pre>
                                            template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type =
04168
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<254»; }; // NOLINT
                                            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»; }; //
                           NOLINT
04170
                                            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<205, ZPZV<254»;
                           }; // 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<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<12>, ZPZV<225>, ZPZV<20>, ZPZV<20
                            ZPZV<3»; }; // NOLINT</pre>
04172
                                          template<> struct ConwayPolynomial<257, 11> { using ZPZ = aerobus::zpz<257>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<40>, ZPZV<254»; };</pre>
                                                                                                                                      // NOLINT
                                          template<> struct ConwayPolynomial<257, 12> { using ZPZ = aerobus::zpz<257>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<13>, ZPZV<225>, ZPZV<215>, ZPZV<173>,
                           ZPZV<249>, ZPZV<148>, ZPZV<20>, ZPZV<3»; }; // NOLINT
  template<> struct ConwayPolynomial<257, 13> { using ZPZ = aerobus::zpz<257>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
```

```
ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<254»; };</pre>
                            template<> struct ConwayPolynomial<257, 17> { 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<0>, ZPZV<8>, ZPZV<254»; }; // NOLINT</pre>
                 template<> struct ConwayPolynomial<257, 19> { using ZPZ = aerobus::zpz<257>; using type = POLYV<2PZV<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<10>, ZPZV<254»; }; //</pre>
                             template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
04177
                POLYV<ZPZV<1>, ZPZV<258»; }; // NOLINT
                            template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
04178
                 POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; }; // NOLINT
04179
                             template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; }; // NOLINT
 04180
                           template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
04181
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<258*; }; // NOLINT template<> struct ConwayPolynomial<263, 6> { using ZPZ = aerobus::zpz<263>; using type =
                 POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<250>, ZPZV<225>, ZPZV<5»; }; // NOLINT
                             template<> struct ConwayPolynomial<263, 7> { using ZPZ = aerobus::zpz<263>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<258»; }; // NOLINT
04184
                           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
                             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<0>, ZPZV<26>, ZPZV<261>, ZPZV<29>, ZPZV<258»;
                 }; // NOLINT
04186
                             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
04187
                             template<> struct ConwayPolynomial<263, 11> { using ZPZ = aerobus::zpz<263>; using type
                 POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                 ZPZV<2>, ZPZV<258»; }; // NOLINT</pre>
                 \label{eq:convergence} template<> struct ConwayPolynomial<263, 12> \{ using ZPZ = aerobus::zpz<263>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<172>, ZPZV<174>, ZPZV<162>, ZPZV<252>, ZPZV<252
04188
                 ZPZV<47>, ZPZV<45>, ZPZV<180>, ZPZV<5»; }; // NOLINT
    template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
                POLYV<ZPZV<1>, ZPZV<267»; }; // NOLINT
                            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 =
04191
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<267»; }; // NOLINT template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
 04192
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2»; }; // NOLINT
 04193
                             template<> struct ConwayPolynomial<269, 5> { using ZPZ = aerobus::zpz<269>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<267»; }; // NOLINT
04194
                            template<> struct ConwayPolynomial<269, 6> { using ZPZ = aerobus::zpz<269>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<120>, ZPZV<101>, ZPZV<206>, ZPZV<2%; }; // NOLINT template<> struct ConwayPolynomial<269, 7> { using ZPZ = aerobus::zpz<269>; using type =
04195
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<66, ZPZV<267»; };
                            template<> struct ConwayPolynomial<269, 8> { using ZPZ = aerobus::zpz<269>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<220>, ZPZV<131>, ZPZV<232>, ZPZV<2»; }; //
04197
                            template<> struct ConwayPolynomial<269, 9> { using ZPZ = aerobus::zpz<269>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2+, ZPZV<214>, ZPZV<267>, ZPZV<267>;
 04198
                             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
   template<> struct ConwayPolynomial<269, 11> { using ZPZ = aerobus::zpz<269>; using type =
04199
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                 ZPZV<20>, ZPZV<267»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<165>, ZPZV<663>, ZPZV<215>,
                 ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2»; }; // NOLINT</pre>
04201
                             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 =
04202
                 POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; }; // NOLINT
                             template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
 04203
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type =
 04204
                 POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<271, 5> { using ZPZ = aerobus::zpz<271>; using type =
04205
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT
                            template<> struct ConwayPolynomial<271, 6> { using ZPZ = aerobus::zpz<271>; using type =
 04206
                 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 =
04207
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<26>; }; // NOLINT template<> struct ConwayPolynomial<271, 8> { using ZPZ = aerobus::zpz<271>; using type =
 04208
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<114>, ZPZV<69>, ZPZV<6»; }; //
 04209
                           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<26>, ZPZV<26>, ZPZV<186>, ZPZV<186>, ZPZV<265»;
                 }; // NOLINT
 04210
                           template<> struct ConwayPolynomial<271, 10> { using ZPZ = aerobus::zpz<271>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<133>, ZPZV<10>, ZPZV<256>, ZPZV<74>,
               ZPZV<126>, ZPZV<6»; }; // NOLINT</pre>
04211
                         template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<10>, ZPZV<265»; }; // NOLINT</pre>
04212
                         template<> struct ConwayPolynomial<271, 12> { using ZPZ = aerobus::zpz<271>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<2162>, ZPZV<210>, ZPZV<116>, ZPZV<205>,
                ZPZV<237>, ZPZV<256>, ZPZV<130>, ZPZV<6»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
04213
               POLYV<ZPZV<1>, ZPZV<272»; }; // NOLINT
                         template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
04214
               POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; }; // NOLINT
                         template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
04215
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; }; // NOLINT
04216
                        template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<22>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<277, 5> { using ZPZ = aerobus::zpz<277>; using type =
04217
              POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<9>, ZPZV<118>, ZPZV<5»; }; // NOLINT
                          template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<272»; }; // NOLINT
04220
                        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 =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177>, ZPZV<110>, ZPZV<272»;
               }; // NOLINT
04222
                          template<> struct ConwayPolynomial<277, 10> { using ZPZ = aerobus::zpz<277>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<206>, ZPZV<253>, ZPZV<237>, ZPZV<241>, ZPZV<260>, ZPZV<5»; }; // NOLINT
                          template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type
               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                ZPZV<5>, ZPZV<272»; }; // NOLINT</pre>
               \label{eq:convergence} 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>, ZPZV<40>, ZPZV<40>, ZPZV<50>, ZPZV<50
04224
               ZPZV<180>, ZPZV<115>, ZPZV<202>, ZPZV<5»; }; // NOLINT
  template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<278»; }; // NOLINT
                        template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
              POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
04227
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<278»; }; // NOLINT template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
04228
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3»; }; // NOLINT
04229
                         template<> struct ConwayPolynomial<281, 5> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<278»; }; // NOLINT
04230
                        template<> struct ConwayPolynomial<281, 6> { using ZPZ = aerobus::zpz<281>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<151>, ZPZV<13>, ZPZV<27>, ZPZV<28>; ising type = template<> struct ConwayPolynomial<281, 7> { using ZPZ = aerobus::zpz<281>; using type = template<> struct ConwayPolynomial<281, 7> { using ZPZ = aerobus::zpz<281>; using type = template
04231
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<278»; }; // NOLINT
                         template<> struct ConwayPolynomial<281, 8> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195>, ZPZV<279>, ZPZV<140>, ZPZV<3»; }; //
                        template<> struct ConwayPolynomial<281, 9> { using ZPZ = aerobus::zpz<281>; using type =
04233
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<70>, ZPZV<278»;
                          template<> struct ConwayPolynomial<281, 10> { using ZPZ = aerobus::zpz<281>; using type =
04234
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<145>, ZPZV<13>, ZPZV<138>,
               ZPZV<191>, ZPZV<3»; }; // NOLINT
  template<> struct ConwayPolynomial<281, 11> { using ZPZ = aerobus::zpz<281>; using type =
04235
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<36>, ZPZV<278»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<2PZV<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
template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
04237
               POLYV<ZPZV<1>, ZPZV<280»; }; // NOLINT
                         template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
04238
               POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3»; }; // NOLINT
                         template<> struct ConwayPolynomial<283, 3> { using ZPZ = aerobus::zpz<283>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type =
04240
               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
04241
               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 =
04242
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<68>, ZPZV<73>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type =
04243
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28, ZPZV<280»; }; // NOLINT template<> struct ConwayPolynomial<283, 8> { using ZPZ = aerobus::zpz<283>; using type =
04244
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<179>, ZPZV<32>, ZPZV<232>, ZPZV<23»; }; //
04245
                        template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<65>, ZPZV<280»;
               }; // NOLINT
04246
                       template<> struct ConwayPolynomial<283, 10> { using ZPZ = aerobus::zpz<283>; using type =
```

```
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 =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
              ZPZV<4>, ZPZV<280»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<283, 12> { using ZPZ = aerobus::zpz<283>; using type =
04248
             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20, ZPZV<20>, ZPZV<20>, ZPZV<20>, ZPZV<49>, ZPZV<49, ZPZV<14>, ZPZV<56>, ZPZV<56>, ZPZV<3»; }; // NOLINT
                       template<> struct ConwayPolynomial<293, 1> { using ZPZ = aerobus::zpz<293>; using type =
04249
             POLYV<ZPZV<1>, ZPZV<291»; }; // NOLINT
                       template<> struct ConwayPolynomial<293, 2> { using ZPZ = aerobus::zpz<293>; using type =
04250
             POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; }; // NOLINT
                       template<> struct ConwayPolynomial<293, 3> { using ZPZ = aerobus::zpz<293>; using type =
04251
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
04252
                      template<> struct ConwayPolynomial<293, 4> { using ZPZ = aerobus::zpz<293>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
04253
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
                        template<> struct ConwayPolynomial<293, 6> { using ZPZ = aerobus::zpz<293>; using type =
             POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<210>, ZPZV<260>, ZPZV<2»; }; // 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»; }; // NOLINT
04256
                      template<> struct ConwayPolynomial<293, 8> { using ZPZ = aerobus::zpz<293>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<2*; }; //
             NOLINT
                        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»;
              }; // NOLINT
04258
                        template<> struct ConwayPolynomial<293, 10> { using ZPZ = aerobus::zpz<293>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<186>, ZPZV<28>, ZPZV<46>, ZPZV<46>, ZPZV<24>,
              ZPZV<2»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<293, 11> { using ZPZ = aerobus::zpz<293>; using type
              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
              ZPZV<3>, ZPZV<291»; }; // NOLINT</pre>
             \label{eq:convergence} template<> struct ConwayPolynomial<293, 12> \{ using ZPZ = aerobus::zpz<293>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<159>, ZPZV<210>, ZPZV<215>, ZPZV<125>, ZPZV<212>, ZPZV<159>, ZPZV<210>, ZPZV<125>, ZPZV<212>, ZPZV<212
04260
             ZPZV<167>, ZPZV<144>, ZPZV<157>, ZPZV<2»; }; // NOLINT
   template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
             POLYV<ZPZV<1>, ZPZV<302»; }; // NOLINT
                      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 =
04263
             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<302»; }; // NOLINT template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
04264
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5»; }; // NOLINT
04265
                        template<> struct ConwayPolynomial<307, 5> { using ZPZ = aerobus::zpz<307>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<302»; }; // NOLINT
04266
                      template<> struct ConwayPolynomial<307, 6> { using ZPZ = aerobus::zpz<307>; using type =
             POLYV<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 = aerobus::zpz<3
04267
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<302»; };
                       template<> struct ConwayPolynomial<307, 8> { using ZPZ = aerobus::zpz<307>; using type
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<283>, ZPZV<232>, ZPZV<131>, ZPZV<5»; }; //
                      template<> struct ConwayPolynomial<307, 9> { using ZPZ = aerobus::zpz<307>; using type =
04269
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<165>, ZPZV<70>, ZPZV<302»;
             }; // NOLINT
   template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
04270
             POLYV<ZPZV<1>, ZPZV<294»; }; // NOLINT
                        template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
04271
             POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; }; // NOLINT
                      template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
04272
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; }; // NOLINT
                        template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; }; // NOLINT
04274
                      template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
             04275
                      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
04276
                        template<> struct ConwayPolynomial<311, 7> { using ZPZ = aerobus::zpz<311>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<294»; }; // NOLI template<> struct ConwayPolynomial<311, 8> { using ZPZ = aerobus::zpz<311>; using type =
04277
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<162>, ZPZV<118>, ZPZV<2>, ZPZV<27»; }; //
              NOLINT
                      template<> struct ConwayPolynomial<311, 9> { using ZPZ = aerobus::zpz<311>; using type
04278
              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<287>, ZPZV<74>, ZPZV<294»;
              }; // NOLINT
04279
                       template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
             POLYV<ZPZV<1>, ZPZV<303»; }; // NOLINT
                       template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
04280
             POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; }; // NOLINT
                       template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; }; // NOLINT
04282
                      template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<239>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
04283
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<303»; }; // NOLINT
```

```
04284
                    template<> struct ConwayPolynomial<313, 6> { using ZPZ = aerobus::zpz<313>; using type =
            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<196>, ZPZV<213>, ZPZV<253>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<313, 7> { using ZPZ = aerobus::zpz<313>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<3033»; }; // NOLINT template<> struct ConwayPolynomial<313, 8> { using ZPZ = aerobus::zpz<313>; using type =
04286
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<306>, ZPZV<99>, ZPZV<106>, ZPZV<10»; }; //
                    template<> struct ConwayPolynomial<313, 9> { using ZPZ = aerobus::zpz<313>; using type
04287
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<267>, ZPZV<300>, ZPZV<303»;
            }; // NOLINT
04288
                    template<> struct ConwayPolynomial<317, 1> { using ZPZ = aerobus::zpz<317>; using type =
            POLYV<ZPZV<1>, ZPZV<315»; }; // NOLINT
                    template<> struct ConwayPolynomial<317, 2> { using ZPZ = aerobus::zpz<317>; using type =
04289
            POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; }; // NOLINT
04290
                   template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<315»; ); // NOLINT
template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<317, 5> { using ZPZ = aerobus::zpz<317>; using type =
04291
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<315»; // NOLINT
                    template<> struct ConwayPolynomial<317, 6> { using ZPZ = aerobus::zpz<317>; using type =
04293
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<195>, ZPZV<156>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                   template<> struct ConwayPolynomial<317, 7> { using ZPZ = aerobus::zpz<317>; using type =
04294
           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
04295
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<207>, ZPZV<85>, ZPZV<31>, ZPZV<22; };
            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<284>, ZPZV<296>, ZPZV<315»;</pre>
04296
            }; // NOLINT
                    template<> struct ConwayPolynomial<331, 1> { using ZPZ = aerobus::zpz<331>; using type =
04297
            POLYV<ZPZV<1>, ZPZV<328»; }; // NOLINT
                    template<> struct ConwayPolynomial<331, 2> { using ZPZ = aerobus::zpz<331>; using type =
            POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; }; // NOLINT
04299
                    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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<290>, ZPZV<3»; }; // NOLINT
04300
04301
                    template<> struct ConwayPolynomial<331, 5> { using ZPZ = aerobus::zpz<331>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<328»; }; // NOLINT
            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
template<> struct ConwayPolynomial<331, 7> { using ZPZ = aerobus::zpz<331>; using type =
04302
04303
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                    template<> struct ConwayPolynomial<331, 8> { using ZPZ = aerobus::zpz<331>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<249>, ZPZV<308>, ZPZV<78>, ZPZV<3»; };
            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<194>, ZPZV<210>, ZPZV<328»;</pre>
04305
            }; // NOLINT
04306
                    template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
            POLYV<ZPZV<1>, ZPZV<327»; }; // NOLINT
04307
                    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 =
04308
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT
                    template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10»; }; // NOLINT
                    template<> struct ConwayPolynomial<337, 5> { using ZPZ = aerobus::zpz<337>; using type =
04310
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT
                   template<> struct ConwayPolynomial<337, 6> { using ZPZ = aerobus::zpz<337>; using type =
04311
            POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<216>, ZPZV<109, ZPZV<109, ZPZV<109; }; // NOLINT template<> struct ConwayPolynomial<337, 7> { using ZPZ = aerobus::zpz<337>; using type
04312
            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<327»; }; //
04313
                    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
04314
                   template<> struct ConwayPolynomial<337, 9> { using ZPZ = aerobus::zpz<337>; using type =
            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<148>, ZPZV<98>, ZPZV<327»;
            }; // NOLINT
                    template<> struct ConwayPolynomial<347, 1> { using ZPZ = aerobus::zpz<347>; using type =
04315
            POLYV<ZPZV<1>, ZPZV<345»; }; // NOLINT
                    template<> struct ConwayPolynomial<347, 2> { using ZPZ = aerobus::zpz<347>; using type =
04316
            POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; }; // NOLINT
                    template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
04317
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<345»; }; // NOLINT
                    template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
04318
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
04319
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<34>; }; // NOLINT template<> struct ConwayPolynomial<347, 6> { using ZPZ = aerobus::zpz<347>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<343>, ZPZV<26>, ZPZV<56>, ZPZV<2»; }; // NOLINT
04320
                    template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44, ZPZV<345»; };
04322
                   template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<187>, ZPZV<213>, ZPZV<117>, ZPZV<2»; }; //
            NOLTNT
```

```
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<235>, ZPZV<252>, ZPZV<345»;
         }; // NOLINT
04324
              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 =
04325
        POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
04326
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; }; // NOLINT template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
04327
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<349, 5> { using ZPZ = aerobus::zpz<349>; using type =
04328
        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»; }; // NOLINI
04330
               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»; }; // NOLINT template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type =
04331
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<308>, ZPZV<328>, ZPZV<268>, ZPZV<2*; }; //
        template<> struct ConwayPolynomial<349, 9> { using ZPZ = aerobus::zpz<349>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<290>, ZPZV<130>, ZPZV<347»;
04332
         }; // NOLINT
               template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type =
04333
        POLYV<ZPZV<1>, ZPZV<350»; }; // NOLINT
               template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
        POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3»; }; // NOLINT
04335
               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 =
04336
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
04337
        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 =
04338
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<21>, ZPZV<215>, ZPZV<226>, ZPZV<295>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<353, 7> { using ZPZ = aerobus::zpz<353>; using type =
04339
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, 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<1>, ZPZV<182>, ZPZV<26>, ZPZV<37>, ZPZV<3»: };
04341
              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<2>, ZPZV<319>, ZPZV<49>, ZPZV<350»;
         }; // NOLINT
04342
               template<> struct ConwayPolynomial<359, 1> { using ZPZ = aerobus::zpz<359>; using type =
        POLYV<ZPZV<1>, ZPZV<352»; }; // NOLINT
04343
               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 =
04344
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<352»; }; // NOLINT
               template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
04345
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<229>, ZPZV<7»; }; // NOLINT
               template<> struct ConwayPolynomial<359, 5> { using ZPZ = aerobus::zpz<359>; using type =
04346
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; }; // NOLINT
        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 =
04347
04348
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<352»; }; //
              template<> struct ConwayPolynomial<359, 8> { using ZPZ = aerobus::zpz<359>; using type =
04349
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<301>, ZPZV<143>, ZPZV<271>, ZPZV<7»; }; //
         NOLINT
04350
              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<356>, ZPZV<356-, ZPZV<356-, ZPZV<356-, ZPZV<356-, ZPZV<356-, ZPZV<356-, ZPZV<3
         }; // NOLINT
               template<> struct ConwayPolynomial<367, 1> { using ZPZ = aerobus::zpz<367>; using type =
        POLYV<ZPZV<1>, ZPZV<361»; }; // NOLINT
04352
              template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
        POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6»; }; // NOLINT
              template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
04353
        POLYV<ZPZV<1>, ZPZV<1>, ZPZV<10>, ZPZV<361»; }; // NOLINT template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type =
04354
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6»; }; // NOLINT
              template<> struct ConwayPolynomial<367, 5> { using ZPZ = aerobus::zpz<367>; using type =
04355
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<361»; }; // NOLINT template<> struct ConwayPolynomial<367, 6> { using ZPZ = aerobus::zpz<367>; using type =
04356
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<222>, ZPZV<324>, ZPZV<6>; }; // NOLINT template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<361»; };
04358
              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»; }; //
        NOLINT
04359
               template<> struct ConwayPolynomial<367, 9> { using ZPZ = aerobus::zpz<367>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<213>, ZPZV<268>, ZPZV<361»;
         }; // NOLINT
04360
               template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
        POLYV<ZPZV<1>, ZPZV<371»; }; // NOLINT template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
04361
         POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2»; }; // NOLINT
```

```
04362
           template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<371»; }; // NOLINT template<> struct ConwayPolynomial<373, 4> { using ZPZ = aerobus::zpz<373>; using type =
04363
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<373, 5> { using ZPZ = aerobus::zpz<373>; using type =
04364
      POLYV<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 =
       POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<126>, 2PZV<83>, 2PZV<108>, ZPZV<2»; }; // NOLINI
           template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type =
04366
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<371»; }; // NOLINT
04367
           template<> struct ConwayPolynomial<373, 8> { using ZPZ = aerobus::zpz<373>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<203>, ZPZV<219>, ZPZV<66>, ZPZV<2»; }; //
           template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type =
04368
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<238>, ZPZV<370>, ZPZV<371»;
       }; // NOLINT
           template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
04369
      POLYV<ZPZV<1>, ZPZV<377»; }; // NOLINT
           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 =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377»; }; // NOLINT template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
04372
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<379, 5> { using ZPZ = aerobus::zpz<379>; using type =
04373
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<377»; }; // NOLINT
04374
           template<> struct ConwayPolynomial<379, 6> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<374>, ZPZV<364>, ZPZV<246>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<379, 7> { using ZPZ = aerobus::zpz<379>; using type
04375
      POLYV<ZPZV<1>, 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 =
04376
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<210>, ZPZV<194>, ZPZV<173>, ZPZV<2»; }; //
           template<> struct ConwayPolynomial<379, 9> { using ZPZ = aerobus::zpz<379>; using type =
04377
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<362>, ZPZV<369>, ZPZV<377»;
       }; // NOLINT
      template<> struct ConwayPolynomial<383, 1> { using ZPZ = aerobus::zpz<383>; using type = POLYV<ZPZV<1>, ZPZV<378»; }; // NOLINT
04378
            template<> struct ConwayPolynomial<383, 2> { using ZPZ = aerobus::zpz<383>; using type =
       POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; }; // NOLINT
04380
           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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5»; }; // NOLINT
04381
           template<> struct ConwayPolynomial<383, 5> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
04383
           template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<8>, ZPZV<158>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 7> { using ZPZ = aerobus::zpz<383>; using type =
04384
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<378»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 8> { using ZPZ = aerobus::zpz<383>; using type
04385
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<281>, ZPZV<332>, ZPZV<296>, ZPZV<5»; }; //
       NOLINT
      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<137>, ZPZV<76>, ZPZV<378»;</pre>
04386
       }; // NOLINT
04387
            template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
       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
           template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
04389
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387»; }; // NOLINT template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
04390
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<266>, ZPZV<2»; }; // NOLINT
04391
           template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<387»; }; // NOLINT
04392
           template<> struct ConwayPolynomial<389, 6> { using ZPZ = aerobus::zpz<389>; using type =
      POLYY<ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<218>, ZPZV<255>, ZPZV<25; // NOLINT template<> struct ConwayPolynomial<389, 7> { using ZPZ = aerobus::zpz<389>; using type
04393
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<387»; }; //
           template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::zpz<389>; using type
04394
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<351>, ZPZV<19>, ZPZV<290>, ZPZV<2»; };
       NOLINT
04395
           template<> struct ConwayPolynomial<389, 9> { using ZPZ = aerobus::zpz<389>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<308>, ZPZV<387»;
       }; // NOLINT
04396
           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 =
04397
      POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<397, 3> { using ZPZ = aerobus::zpz<397>; using type =
04398
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<392»; }; // NOLINT
           template<> struct ConwayPolynomial<397, 4> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5»; }; // NOLINT
      template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392»; }; // NOLINT</pre>
04400
04401
           template<> struct ConwayPolynomial<397, 6> { using ZPZ = aerobus::zpz<397>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<274>, ZPZV<287>, ZPZV<5»; };
               template<> struct ConwayPolynomial<397, 7> { using ZPZ = aerobus::zpz<397>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<392»; };
                                                                                                                                       // NOLINT
             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»; }; //
         NOLTNT
04404
              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<166>, ZPZV<252>, ZPZV<392»;
         }; // NOLINT
04405
              template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
        POLYV<ZPZV<1>, ZPZV<398»; }; // NOLINT
              \texttt{template<>} \texttt{struct ConwayPolynomial<401, 2> \{ \texttt{using ZPZ = aerobus::zpz<401>; using type = 1.5 \}}
04406
        POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<398»; }; // NOLINT
              template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type =
04408
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<37; }; // NOLINT template<> struct ConwayPolynomial<401, 5> { using ZPZ = aerobus::zpz<401>; using type =
04409
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<398»; }; // NOLINT
              template<> struct ConwayPolynomial<401, 6> { using ZPZ = aerobus::zpz<401>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<115>, ZPZV<61>, ZPZV<51>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<401, 7> { using ZPZ = aerobus::zpz<401>; using type =
04411
        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 =
04412
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<380>, ZPZV<113>, ZPZV<164>, ZPZV<3»; }; //
              template<> struct ConwayPolynomial<401, 9> { using ZPZ = aerobus::zpz<401>; using type =
04413
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<158>, ZPZV<398»;
         }; // NOLINT
04414
              template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
        POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
04415
               template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
         POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT
04416
              template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388»; }; // NOLINT template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
04417
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<12>, ZPZV<407>, ZPZV<21»; }; // NOLINT template<> struct ConwayPolynomial<409, 5> { using ZPZ = aerobus::zpz<409>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<388»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 6> { using ZPZ = aerobus::zpz<409>; using type =
04419
        POLYV<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 =
04420
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<388»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
04421
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<256>, ZPZV<69>, ZPZV<396>, ZPZV<21»; }; //
         NOLINT
04422
              template<> struct ConwayPolynomial<409, 9> { using ZPZ = aerobus::zpz<409>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<318>, ZPZV<211>, ZPZV<388»;
         }; // NOLINT
04423
               template<> struct ConwavPolynomial<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 =
         POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
04425
        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
04426
              template<> struct ConwayPolynomial<419, 5> { using ZPZ = aerobus::zpz<419>; using type =
04427
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; // NOLINT
04428
               template<> struct ConwayPolynomial<419, 6> { using ZPZ = aerobus::zpz<419>; using type =
        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 = aerobus::zpz<4
04429
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; // NOLINT
               template<> struct ConwayPolynomial<419, 8> { using ZPZ = aerobus::zpz<419>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<38>, ZPZV<388>, ZPZV<151>, ZPZV<2»; }; //
         NOLINT
04431
        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<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3</pre>
         }; // NOLINT
04432
               template<> struct ConwayPolynomial<421, 1> { using ZPZ = aerobus::zpz<421>; using type =
        POLYV<ZPZV<1>, ZPZV<419»; }; // NOLINT
04433
              template<> struct ConwayPolynomial<421, 2> { using ZPZ = aerobus::zpz<421>; using type =
        POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
04434
        POLYY<ZPZY<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
        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
04436
04437
              template<> struct ConwayPolynomial<421, 6> { using ZPZ = aerobus::zpz<421>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<111>, ZPZV<342>, ZPZV<41>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<421,
                                                                         7> { using ZPZ = aerobus::zpz<421>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<419»; }; //
04439
              template<> struct ConwayPolynomial<421, 8> { using ZPZ = aerobus::zpz<421>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<389>, ZPZV<32>, ZPZV<77>, ZPZV<2»; };
04440
              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»;
04441
          template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
04442
          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 =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424»; }; // NOLINT
          template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
04444
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<431, 5> { using ZPZ = aerobus::zpz<431>; using type =
04445
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<424»; }; // NOLINT
04446
          template<> struct ConwayPolynomial<431, 6> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<161>, ZPZV<202>, ZPZV<182>, ZPZV<7»; }; // NOLIN
04447
          template<> struct ConwayPolynomial<431, 7> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT template<> struct ConwayPolynomial<431, 8> { using ZPZ = aerobus::zpz<431>; using type =
04448
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<543>, ZPZV<286>, ZPZV<115>, ZPZV<7»; }; //
          template<> struct ConwayPolynomial<431, 9> { using ZPZ = aerobus::zpz<431>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<71>, ZPZV<329>, ZPZV<424*;
      }; // NOLINT
04450
          template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<428»; }; // NOLINT
04451
          template<> struct ConwayPolynomial<433, 2> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5»; }; // NOLINT
          template<> struct ConwayPolynomial<433, 3> { using ZPZ = aerobus::zpz<433>; using type =
04452
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428»; }; // NOLINT template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
04453
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<402>, ZPZV<5»; }; // NOLINT

template<> struct ConwayPolynomial<433, 5> { using ZPZ = aerobus::zpz<433>; using type =
04454
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<428»; }; // NOLINT
          template<> struct ConwayPolynomial<433, 6> { using ZPZ = aerobus::zpz<433>; using type =
04455
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<244>, ZPZV<353>, ZPZV<360>, ZPZV<5»; }; // NOLINT
04456
          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<0>, ZPZV<6>, ZPZV<6>, ZPZV<428»; }; // NOLINT
          template<> struct ConwayPolynomial<433, 8> { using ZPZ = aerobus::zpz<433>; using type =
04457
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347>, ZPZV<32>, ZPZV<39>, ZPZV<5»; };
          template<> struct ConwayPolynomial<433, 9> { using ZPZ = aerobus::zpz<433>; using type =
04458
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<232>, ZPZV<45>, ZPZV<428»;
      }; // NOLINT
          template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
04459
      POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
          template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; }; // NOLINT
04461
          template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424»; }; // NOLINT template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
04462
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; }; // NOLINT
          template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
04463
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT
04464
          template<> struct ConwayPolynomial<439, 6> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<324>, ZPZV<190>, ZPZV<15»; }; // NOLINT
          template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type =
04465
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT
          template<> struct ConwayPolynomial<439, 8> { using ZPZ = aerobus::zpz<439>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<359>, ZPZV<296>, ZPZV<266>, ZPZV<15»; }; //
04467
          template<> struct ConwayPolynomial<439, 9> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<424»;
      }; // NOLINT
04468
           template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<441»; }; // NOLINT
04469
          template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; }; // NOLINT
04470
          template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2»; }; // NOLINT
04472
          template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT
04473
          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
04474
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<441»; }; //
          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»; }; //
      NOLINT
04476
      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<3>, ZPZV<125>, ZPZV<105>, ZPZV<441»;</pre>
      }; // NOLINT
    template<> struct ConwayPolynomial<449, 1> { using ZPZ = aerobus::zpz<449>; using type =
     POLYV<ZPZV<1>, ZPZV<446»; }; // NOLINT
          template<> struct ConwayPolynomial<449, 2> { using ZPZ = aerobus::zpz<449>; using type =
04478
     POLYV<ZPZV<1>, ZPZV<444>, ZPZV<3»; }; // NOLINT
          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 =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3»; }; // NOLINT
                 template<> struct ConwayPolynomial<449, 5> { using ZPZ = aerobus::zpz<449>; using type =
04481
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<446»; }; // NOLINT
04482
                 template<> struct ConwayPolynomial<449, 6> { using ZPZ = aerobus::zpz<449>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<293>, ZPZV<69>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<449, 7> { using ZPZ = aerobus::zpz<449>; using type =
04483
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<446»; }; // NOLI template<> struct ConwayPolynomial<449, 8> { using ZPZ = aerobus::zpz<449>; using type =
04484
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<348>, ZPZV<124>, ZPZV<3»; }; //
          NOLINT
                 template<> struct ConwayPolynomial<449, 9> { using ZPZ = aerobus::zpz<449>; using type =
04485
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<226>, ZPZV<29>, ZPZV<4446»; };
           // NOLINT
                 template<> struct ConwayPolynomial<457, 1> { using ZPZ = aerobus::zpz<457>; using type =
04486
          POLYV<ZPZV<1>, ZPZV<444»; }; // NOLINT
                 template<> struct ConwayPolynomial<457, 2> { using ZPZ = aerobus::zpz<457>; using type =
04487
          POLYV<ZPZV<1>, ZPZV<454>, ZPZV<13»; }; // NOLINT
                 template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<4444»; }; // NOLINT template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
04489
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<457, 5> { using ZPZ = aerobus::zpz<457>; using type =
04490
          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 =
04491
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<205>, ZPZV<389>, ZPZV<266>, ZPZV<13»; }; // NOLINT
04492
                 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»; };
                 template<> struct ConwayPolynomial<457, 8> { using ZPZ = aerobus::zpz<457>; using type =
04493
          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 =
04494
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<444*;
          }; // NOLINT
04495
                  template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
          POLYV<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 =
04497
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459»; }; // NOLINT template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
04498
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
04499
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<459»; }; // NOLINT
04500
                  template<> struct ConwayPolynomial<461, 6> { using ZPZ = aerobus::zpz<461>; using type =
           \verb"Polyv<2pzv<1>, & 2pzv<0>, & 2pzv<1>, & 2pzv<439>, & 2pzv<432>, & 2pzv<329>, & 2pzv<2»; & \}; & // & Nolint & (Apzv) 
04501
                 template<> struct ConwayPolynomial<461, 7> { using ZPZ = aerobus::zpz<461>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<459»; }; // NOLINT
                 template<> struct ConwayPolynomial<461, 8> { using ZPZ = aerobus::zpz<461>; using type =
04502
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<449>, ZPZV<321>, ZPZV<2»; }; //
04503
                 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<276>, ZPZV<459»;
          }; // NOLINT
                  template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
04504
          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
04506
                  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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; }; // NOLINT
04507
                  template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<460»; }; // NOLINT
04509
                 template<> struct ConwayPolynomial<463, 6> { using ZPZ = aerobus::zpz<463>; using type =
         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 =
04510
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<460»; }; // NOLINT
                  template<> struct ConwayPolynomial<463, 8> { using ZPZ = aerobus::zpz<463>; using type =
04511
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<234>, ZPZV<414>, ZPZV<396>, ZPZV<39; }; //
          \label{eq:convergence} $$ \text{template<> struct ConwayPolynomial<463, 9> { using ZPZ = aerobus::zpz<463>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<433>, ZPZV<427>, ZPZV<227>, ZPZV<460»; ZPZV<0>, ZPZV
04512
          }; // NOLINT
04513
                  template<> struct ConwayPolynomial<467, 1> { using ZPZ = aerobus::zpz<467>; using type =
          POLYV<ZPZV<1>, ZPZV<465»; }; // NOLINT
04514
                 template<> struct ConwayPolynomial<467, 2> { using ZPZ = aerobus::zpz<467>; using type =
          POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
04515
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
                 template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
04516
          POLYV<ZPZV<1>, ZPZV<3>, ZPZV<14>, ZPZV<353>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<467, 5> { using ZPZ = aerobus::zpz<467>; using type =
04517
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 6> { using ZPZ = aerobus::zpz<467>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<2>; }; // NOLINT
04518
```

```
template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<467, 8> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<318>, ZPZV<413>, ZPZV<289>, ZPZV<2»; }; //
      NOLINT
           template<> struct ConwayPolynomial<467, 9> { using ZPZ = aerobus::zpz<467>; using type =
04521
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<397>, ZPZV<4447>, ZPZV<465»;
      }; // NOLINT
04522
           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 =
04523
      POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; }; // NOLINT
           template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
04524
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466»; }; // NOLINT
04525
          template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<479, 5> { using ZPZ = aerobus::zpz<479>; using type =
04526
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20, ZPZV<2>, ZPZV<466s; }; // NOLINT template<> struct ConwayPolynomial<479, 6> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<243>, ZPZV<287>, ZPZV<334>, ZPZV<13»; }; // 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<4>, ZPZV<46»; }; // NOLINT
04529
          template<> struct ConwayPolynomial<479, 8> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<247>, ZPZV<440>, ZPZV<17>, ZPZV<13»; }; //
      NOLINT
           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<185>, ZPZV<466»; };
      // NOLINT
04531
           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 =
04532
      POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type =
04534
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
04535
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<484»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 6> { using ZPZ = aerobus::zpz<487>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<427>, ZPZV<185>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<487, 7> { using ZPZ = aerobus::zpz<487>; using type =
04537
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, 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 =
04538
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<249>, ZPZV<137>, ZPZV<3»; }; //
04539
           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<10>, ZPZV<21>, ZPZV<271>, ZPZV<447>, ZPZV<484»;
      }; // NOLINT
04540
           template<> struct ConwavPolynomial<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 =
04541
      POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; }; // NOLINT
04542
           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 =
04543
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360>, ZPZV<2%; }; // NOLINT template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 6> { using ZPZ = aerobus::zpz<491>; using type =
04545
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<369>, ZPZV<402>, ZPZV<125>, ZPZV<2»; }; // NOLINT
04546
           template<> struct ConwayPolynomial<491, 7> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 8> { using ZPZ = aerobus::zpz<491>, using type =
      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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<453>, ZPZV<453>, ZPZV<4453>,
04548
      }; // NOLINT
           template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
04549
      POLYV<ZPZV<1>, ZPZV<492»; }; // NOLINT
           template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
04550
      POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
04551
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
04552
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
04553
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<492»; }; // NOLINT
04554
           template<> struct ConwayPolynomial<499, 6> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<407>, ZPZV<191>, ZPZV<78>, ZPZV<78>; // NOLINT template<> struct ConwayPolynomial<499, 7> { using ZPZ = aerobus::zpz<499>; using type =
04555
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<492»; };
           template<> struct ConwayPolynomial<499, 8> { using ZPZ = aerobus::zpz<499>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<288>, ZPZV<309>, ZPZV<200>, ZPZV<7»; }; //
      template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<491>, ZPZV<222>, ZPZV<492»;</pre>
```

```
}; // NOLINT
04558
               template<> struct ConwayPolynomial<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 =
04559
        POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
04560
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<498»; }; // NOLINT
               template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type =
04561
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<325>, ZPZV<5*; }; // NOLINT
template<> struct ConwayPolynomial<503, 5> { using ZPZ = aerobus::zpz<503>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<498»; }; // NOLINT
template<> struct ConwayPolynomial<503, 6> { using ZPZ = aerobus::zpz<503>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
04562
04563
        POLYY<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<380>, ZPZV<292>, ZPZV<255>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<503, 7> { using ZPZ = aerobus::zpz<503>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<498»; };
04565
              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
04566
              template<> struct ConwayPolynomial<503, 9> { using ZPZ = aerobus::zpz<503>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<158>, ZPZV<337>, ZPZV<498»;
        }; // NOLINT
04567
               template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
        POLYV<ZPZV<1>, ZPZV<507»; }; // NOLINT
              template<> struct ConwayPolynomial<509, 2> { using ZPZ = aerobus::zpz<509>; using type =
04568
        POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
04570
        template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
04571
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
04572
               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»; }; // NOLINI
04573
             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<6>, ZPZV<6>, ZPZV<507»; }; // NOLINT template<> struct ConwayPolynomial<509, 8> { using ZPZ = aerobus::zpz<509>; using type =
04574
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<420>, ZPZV<473>, ZPZV<382>, ZPZV<2»; }; //
               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<0>, ZPZV<314>, ZPZV<28>, ZPZV<507»;
        }; // NOLINT
04576
              template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
        POLYV<ZPZV<1>, ZPZV<518»; }; // NOLINT
              template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
04577
        POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; }; // NOLINT
04578
               template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<51a>; }; // NOLINT
template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<509>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<521, 5> { using ZPZ = aerobus::zpz<521>; using type =
04579
04580
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<518»; }; // NOLINT
               template<> struct ConwayPolynomial<521, 6> { using ZPZ = aerobus::zpz<521>; using type =
04581
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<315>, ZPZV<153>, ZPZV<280>, ZPZV<3»; }; // NOLINT
04582
              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<1>, ZPZV<518»; }; // NOLINT
              template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type =
04583
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<46>, ZPZV<407>, ZPZV<312>, ZPZV<31x; //
        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<181>, ZPZV<483>, ZPZV<483>, ZPZV<518»;
        }; // NOLINT
04585
               template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
        POLYV<ZPZV<1>, ZPZV<521»; }; // NOLINT
               template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
        POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; }; // NOLINT
04587
              template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
04588
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
04589
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<521»; }; // NOLINT
04590
             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»; }; // NOLINT template<> struct ConwayPolynomial<523, 7> { using ZPZ = aerobus::zpz<523>; using type =
04591
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<13>, ZPZV<521»; };
                                                                                                                                     // NOLINT
               template<> struct ConwayPolynomial<523, 8> { using ZPZ = aerobus::zpz<523>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<518>, ZPZV<184>, ZPZV<380>, ZPZV<2»; };
04593
              template<> struct ConwayPolynomial<523, 9> { using ZPZ = aerobus::zpz<523>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<342>, ZPZV<145>, ZPZV<521»;
         }; // NOLINT
               template<> struct ConwayPolynomial<541, 1> { using ZPZ = aerobus::zpz<541>; using type =
        POLYV<ZPZV<1>, ZPZV<539»; }; // NOLINT
04595
              template<> struct ConwayPolynomial<541, 2> { using ZPZ = aerobus::zpz<541>; using type =
        POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
04596
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539»; }; // NOLINT
```

```
04597
               template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
04598
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<539»; }; // NOLINT
        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 =
04599
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<539»; };
              template<> struct ConwayPolynomial<541, 8> { using ZPZ = aerobus::zpz<541>; using type =
04601
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<376>, ZPZV<108>, ZPZV<113>, ZPZV<2»; }; //
         NOLINT
               template<> struct ConwayPolynomial<541, 9> { using ZPZ = aerobus::zpz<541>; using type =
04602
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<340>, ZPZV<316>, ZPZV<316<, ZPZV<316>, ZPZV<316<, ZPZV<316<, ZPZV<316>, ZPZV<316<, ZPZV<31
         }; // NOLINT
04603
               template<> struct ConwayPolynomial<547, 1> { using ZPZ = aerobus::zpz<547>; using type =
         POLYY<ZPZV<1>, ZPZV<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 2> { using ZPZ = aerobus::zpz<547>; using type =
04604
         POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<547, 3> { using ZPZ = aerobus::zpz<547>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 4> { using ZPZ = aerobus::zpz<547>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<334>, ZPZV<2»; }; // NOLINT
04607
              template<> struct ConwayPolynomial<547, 5> { using ZPZ = aerobus::zpz<547>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<545»; }; // NOLINT
04608
               template<> struct ConwayPolynomial<547, 6> { using ZPZ = aerobus::zpz<547>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<42»; }; // NOLINT
04609
              template<> struct ConwayPolynomial<547, 7> { using ZPZ = aerobus::zpz<547>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 8> { using ZPZ = aerobus::zpz<547>; using type =
04610
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2»; }; //
         NOLINT
04611
               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<11>, ZPZV<238>, ZPZV<263>, ZPZV<545»;
         }; // NOLINT
04612
               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 =
04613
         POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; }; // NOLINT
04614
               template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<555»; }; // NOLINT template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type =
04615
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<557, 5> { using ZPZ = aerobus::zpz<557>; using type =
04616
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<555»; }; // NOLINT
               template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<202>, ZPZV<192>, ZPZV<253>, ZPZV<2»; };
                                                                                                                                 // NOLINI
04618
              template<> struct ConwayPolynomial<557, 7> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<555»; }; // NOLINT
04619
              template<> struct ConwayPolynomial<557, 8> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<384>, ZPZV<113>, ZPZV<2»; }; //
         template<> struct ConwayPolynomial<557, 9> { using ZPZ = aerobus::zpz<557>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<456>, ZPZV<434>, ZPZV<434>, ZPZV<555»;
04620
         }; // NOLINT
04621
               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 =
         POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
04623
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; }; // NOLINT template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
04624
        POLYVCZPZV<1>, ZPZV<2), ZPZV<20, ZPZV<399, ZPZV<2; ; // NOLINT
template<> struct ConwayPolynomial<563, 5> { using ZPZ = aerobus::zpz<563>; using type =
04625
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<561»; }; // NOLINT
04626
              template<> struct ConwayPolynomial<563, 6> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<12>, ZPZV<303>, ZPZV<246>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz<563>; using type =
04627
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<561»; }; // NOLINT
              template<> struct ConwayPolynomial<563, 8> { using ZPZ = aerobus::zpz<563>; using type =
04628
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<176>, ZPZV<509>, ZPZV<2»; };
04629
              template<> struct ConwayPolynomial<563, 9> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<15>, ZPZV<15>, ZPZV<19>, ZPZV<561»; };
         // NOLINT
04630
               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 =
04631
         POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; }; // NOLINT
04632
               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 =
04633
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; };
                                                                                               // NOLINT
               template<> struct ConwayPolynomial<569, 5> { using ZPZ = aerobus::zpz<569>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT
04635
              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 template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type =
04636
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<566»; };
                  template<> struct ConwayPolynomial<569, 8> { using ZPZ = aerobus::zpz<569>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<527>, ZPZV<173>, ZPZV<241>, ZPZV<241>, ZPZV<3»; }; //
           NOLINT
04638
                  template<> struct ConwayPolynomial<569, 9> { using ZPZ = aerobus::zpz<569>; using type =
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<478>, ZPZV<566>, ZPZV<566»;
04639
                  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 =
04640
          POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
04641
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<6>, ZPZY<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type =
04642
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3»; }; // NOLINT
04643
                  template<> struct ConwayPolynomial<571, 5> { using ZPZ = aerobus::zpz<571>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<568»; }; // NOLINT
          template<> struct ConwayPolynomial<571, 6> { using ZPZ = aerobus::zpz<571>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<295>, ZPZV<33>, ZPZV<3»; }; // NOLINT
04644
                  template<> struct ConwayPolynomial<571,
                                                                                           7> { using ZPZ = aerobus::zpz<571>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<568»; }; //
04646
                  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), }; //
           NOLINT
04647
                  template<> struct ConwayPolynomial<571, 9> { using ZPZ = aerobus::zpz<571>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<545>, ZPZV<5179>, ZPZV<568»;
           }; // NOLINT
04648
                  template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
          POLYV<ZPZV<1>, ZPZV<572»; }; // NOLINT
                 template<> struct ConwayPolynomial<577, 2> { using ZPZ = aerobus::zpz<577>; using type =
04649
          POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5»; }; // NOLINT
04650
                  template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572»; }; // NOLINT
                 template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
04651
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<577, 5> { using ZPZ = aerobus::zpz<577>; using type =
04652
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<3>, ZPZY<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
                  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<8>, ZPZV<572»; }; // NOLINT template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type =
04655
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<59; }; //
                  template<> struct ConwayPolynomial<577, 9> { using ZPZ = aerobus::zpz<577>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<576>, ZPZV<449>, ZPZV<572»;
           }; // NOLINT
04657
                  template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
          POLYV<ZPZV<1>, ZPZV<585»; }; // NOLINT
                  template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
04658
          POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<587, 3> { using ZPZ = aerobus::zpz<587>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585»; }; // NOLINT template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
04660
          POLYV<ZPZV<1>, ZPZV<1>, ZPZV<16>, ZPZV<444>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<587, 5> { using ZPZ = aerobus::zpz<587>; using type =
04661
           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<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<204>, ZPZV<224>, ZPZV<226>, ZPZV<22*; ); // NOLINT template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type =
04663
          POLYV<ZPZV<1>, ZPZV<0>, 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 =
04664
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<492>, ZPZV<444>, ZPZV<91>, ZPZV<2»; };
                 template<> struct ConwayPolynomial<587, 9> { using ZPZ = aerobus::zpz<587>; using type =
04665
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<333>, ZPZV<55>, ZPZV<585»;
           }; // NOLINT
04666
                  \texttt{template<> struct ConwayPolynomial<593, 1> \{ using ZPZ = aerobus:: zpz<593>; using type = 200 aerobus:: zpz<593>; usin
          POLYV<ZPZV<1>, ZPZV<590»; }; // NOLINT
04667
                  template<> struct ConwayPolynomial<593, 2> { using ZPZ = aerobus::zpz<593>; using type =
          POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3»; }; // NOLINT
04668
                 template<> struct ConwayPolynomial<593, 3> { using ZPZ = aerobus::zpz<593>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<590»; }; // NOLINT template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
04669
          POLYV<ZPZV<1>, ZPZV<4>, ZPZV<419>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<593, 5> { using ZPZ = aerobus::zpz<593>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<590»; }; // NOLINT
04671
                  template<> struct ConwayPolynomial<593, 6> { using ZPZ = aerobus::zpz<593>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<345>, ZPZV<478>, ZPZV<37; }; // NOLINT template<> struct ConwayPolynomial<593, 7> { using ZPZ = aerobus::zpz<593>; using type =
04672
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<590»; }; // NOLINT
                  template<> struct ConwayPolynomial<593, 8> { using ZPZ = aerobus::zpz<593>; using type =
04673
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<350>, ZPZV<291>, ZPZV<495>, ZPZV<3»; };
           NOLINT
04674
                 \texttt{template<>} \texttt{struct ConwayPolynomial<593, 9> \{ \texttt{using ZPZ = aerobus::zpz<593>; 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 + 20
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223>, ZPZV<523>, ZPZV<590»;
           }; // NOLINT
```

```
04675
               template<> struct ConwayPolynomial<599, 1> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<592»; }; // NOLINT
              template<> struct ConwayPolynomial<599, 2> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
04677
               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 =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7»; }; // NOLINT
04679
              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 =
04680
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type
04681
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592»; }; //
04682
             template<> struct ConwayPolynomial<599, 8> { using ZPZ = aerobus::zpz<599>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<440>, ZPZV<37>, ZPZV<124>, ZPZV<7»; };
         NOLTNT
        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<114>, ZPZV<98>, ZPZV<592»;
04683
               template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type =
        POLYV<ZPZV<1>, ZPZV<594»; }; // NOLINT
              template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
04685
        POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
               template<> struct ConwayPolynomial<601, 3> { using ZPZ = aerobus::zpz<601>; using type =
04686
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<594»; }; // NOLINT template<> struct ConwayPolynomial<601, 4> { using ZPZ = aerobus::zpz<601>; using type =
04687
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; }; // NOLINT
               template<> struct ConwayPolynomial<601, 5> { using ZPZ = aerobus::zpz<601>; using type =
04688
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<594»; }; // NOLINT

template<> struct ConwayPolynomial<01, 6> { using ZPZ = aerobus::zpz<601>; using type =
04689
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<440>, ZPZV<49>, ZPZV<7»; };
               template<> struct ConwayPolynomial<601, 7> { using ZPZ = aerobus::zpz<601>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5
              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<7»; }; //
         NOLINT
              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<7>, ZPZV<487>, ZPZV<590>, ZPZV<594»;
         }; // NOLINT
04693
               template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
        POLYV<ZPZV<1>, ZPZV<604»; }; // NOLINT
               template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
04694
        POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<604»; }; // NOLINT
              template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
04696
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<607, 5> { using ZPZ = aerobus::zpz<607>; using type =
04697
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<604»; }; // NOLINT
04698
               template<> struct ConwayPolynomial<607, 6> { using ZPZ = aerobus::zpz<607>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<45>, ZPZV<478>, ZPZV<478); // NOLINT
04699
              template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<604*; }; // NOLINT template<> struct ConwayPolynomial<607, 8> { using ZPZ = aerobus::zpz<607>; using type =
04700
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<468>, ZPZV<35>, ZPZV<449>, ZPZV<3»; };
04701
              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<1>, ZPZV<444>, ZPZV<129>, ZPZV<604»;
         }; // NOLINT
04702
              template<> struct ConwayPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
        POLYV<ZPZV<1>, ZPZV<611»; }; // NOLINT
               template<> struct ConwayPolynomial<613, 2> { using ZPZ = aerobus::zpz<613>; using type =
        POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; }; // NOLINT
04704
              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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
04705
04706
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<611»; }; // NOLINT
04707
               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
04708
               template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, 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 =
04709
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<489>, ZPZV<57>, ZPZV<539>, ZPZV<22»; };
         NOLINT
04710
              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<513>, ZPZV<536>, ZPZV<611»;
         }; // NOLINT
04711
               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 =
        POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; }; // NOLINT
             template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
04713
        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»; };
                template<> struct ConwayPolynomial<617, 5> { using ZPZ = aerobus::zpz<617>; using type =
         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 =
04716
         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
04717
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<614»; }; //
04718
                template<> struct ConwayPolynomial<617, 8> { using ZPZ = aerobus::zpz<617>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, 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»;
04719
         }; // NOLINT
   template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<617»; }; // NOLINT
04721
               template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
04722
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617»; }; // NOLINT
               template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
04723
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492>, ZPZV<2»; }; // NOLINT
04724
               template<> struct ConwayPolynomial<619, 5> { using ZPZ = aerobus::zpz<619>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<617»; // NOLINT template<> struct ConwayPolynomial<619, 6> { using ZPZ = aerobus::zpz<619>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<468>, ZPZV<347>, ZPZV<2»; }; // NOLINT
04725
                template<> struct ConwayPolynomial<619, 7> { using ZPZ = aerobus::zpz<619>; using type
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5
04727
               template<> struct ConwayPolynomial<619, 8> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<416>, ZPZV<383>, ZPZV<225>, ZPZV<2»; }; //
         NOLINT
               template<> struct ConwayPolynomial<619, 9> { using ZPZ = aerobus::zpz<619>; using type =
04728
         POLYV<ZPZV<1>, ZPZV<0>, 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 =
04729
         POLYV<ZPZV<1>, ZPZV<628»; }; // NOLINT
                template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
04730
         POLYV<ZPZV<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 template<> struct ConwayPolynomial<631, 4> { using ZPZ = aerobus::zpz<631>; using type =
04732
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<3»; }; // NOLINT
         template<> struct ConwayPolynomial<asin, 5> { using ZPZ = aerobus::zpz<631>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
04733
04734
               template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<516>, ZPZV<541>, ZPZV<106>, ZPZV<3»; }; // NOLINT
04735
               template<> struct ConwayPolynomial<631, 7> { using ZPZ = aerobus::zpz<631>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<628*; }; // NOLINT template<> struct ConwayPolynomial<631, 8> { using ZPZ = aerobus::zpz<631>; using type =
04736
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<187>, ZPZV<3w; }; //
         NOLINT
04737
               template<> struct ConwayPolynomial<631, 9> { using ZPZ = aerobus::zpz<631>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<296>, ZPZV<413>, ZPZV<628»;
         }; // NOLINT
04738
               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 =
04739
         POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
04740
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<638»; }; // NOLINT
template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type =
04741
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<629>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<641, 5> { using ZPZ = aerobus::zpz<641>; using type =
04742
         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 =
         POLYV<2PZV<1>, 2PZV<0>, 2PZV<2>, 2PZV<105>, ZPZV<557>, ZPZV<294>, ZPZV<3»; }; // NOLINT
04744
               template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<638»; }; // NOLINT template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>; using type =
04745
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<33»; }; //
         NOLINT
               template<> struct ConwayPolynomial<641, 9> { using ZPZ = aerobus::zpz<641>; using type =
04746
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<641>, ZPZV<638»;
         }; // NOLINT
04747
                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 =
         POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11»; }; // NOLINT
04749
               template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<62>; }; // NOLINT
template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<600>, ZPZV<11»; }; // NOLINT
template<> struct ConwayPolynomial<643, 5> { using ZPZ = aerobus::zpz<643>; using type =
04750
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<632»; }; // NOLINT
04752
               template<> struct ConwayPolynomial<643, 6> { using ZPZ = aerobus::zpz<643>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<345>, ZPZV<412>, ZPZV<293>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<643, 7> { using ZPZ = aerobus::zpz<643>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<632»; }; // NOLINT
04753
```

```
template<> struct ConwayPolynomial<643, 8> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<573>, ZPZV<569>, ZPZV<11»; }; //
      NOLINT
      template<> struct ConwayPolynomial<643, 9> { using ZPZ = aerobus::zpz<643>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<591>, ZPZV<591>, ZPZV<475>, ZPZV<632»;</pre>
04755
      }; // NOLINT
           template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<642»; }; // NOLINT
04757
           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 =
04758
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<642»; }; // NOLINT
           template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
04759
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5»; }; // NOLINT
04760
          template<> struct ConwayPolynomial<647, 5> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<642»; }; // NOLINT template<> struct ConwayPolynomial<647, 6> { using ZPZ = aerobus::zpz<647>; using type =
04761
      POLYYCZPZV<1>, ZPZV<5>, ZPZV<50>, ZPZV<308>, ZPZV<642>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<647, 7> { using ZPZ = aerobus::zpz<647>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<642»; };
           template<> struct ConwayPolynomial<647, 8> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<603>, ZPZV<259>, ZPZV<271>, ZPZV<5»; }; //
      NOLINT
      template<> struct ConwayPolynomial<647, 9> { using ZPZ = aerobus::zpz<647>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<561>, ZPZV<123>, ZPZV<561>, ZPZV<123>, ZPZV<642»;</pre>
04764
      }; // NOLINT
04765
           template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<651»; }; // NOLINT
04766
           template<> struct ConwayPolynomial<653, 2> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<653, 3> { using ZPZ = aerobus::zpz<653>; using type =
04767
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<651»; }; // NOLINT template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type =
04768
      04769
           template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651»; }; // NOLINT
           template<> struct ConwayPolynomial<653, 6> { using ZPZ = aerobus::zpz<653>; using type =
04770
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<45>, ZPZV<220>, ZPZV<242>, ZPZV<24>, ZPZV<2, }; // NOLINT
04771
           template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<651»; }; //
04772
          template<> struct ConwayPolynomial<653, 8> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<385>, ZPZV<18>, ZPZV<296>, ZPZV<2»; }; //
      NOLINT
04773
           template<> struct ConwayPolynomial<653, 9> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<365>, ZPZV<6651»;
      }; // NOLINT
04774
           template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<657»; }; // NOLINT
           template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
04775
      POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<657»; }; // NOLINT
04777
           template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
04778
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<657»; }; // NOLINT
           template<> struct ConwayPolynomial<659, 6> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<2PZV<1>, 2PZV<0>, ZPZV<6>, ZPZV<371>, ZPZV<105>, ZPZV<223>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<659, 7> { using ZPZ = aerobus::zpz<659>; using type =
04780
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<657»; }; // NOLIN template<> struct ConwayPolynomial<659, 8> { using ZPZ = aerobus::zpz<659>; using type =
04781
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<358>, ZPZV<246>, ZPZV<90>, ZPZV<2»; }; //
      NOLINT
           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<0>, ZPZV<592>, ZPZV<592>, ZPZV<46>, ZPZV<657»;
      }; // NOLINT
04783
           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 =
04784
      POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
04785
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<659»; }; // NOLINT template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type =
04786
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
04787
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<659»; }; // NOLINT
           template<> struct ConwayPolynomial<661, 6> { using ZPZ = aerobus::zpz<661>; using type =
04788
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<456>, ZPZV<382>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<661, 7> { using ZPZ = aerobus::zpz<661>; using type =
04789
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<659»; }; // NOLINT template<> struct ConwayPolynomial<661, 8> { using ZPZ = aerobus::zpz<661>; using type =
04790
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<612>, ZPZV<285>, ZPZV<72>, ZPZV<2»; }; //
04791
           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<18>, ZPZV<389>, ZPZV<220>, ZPZV<659»;
      }; // NOLINT
04792
          template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<668»; };
               template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
04793
        POLYY<ZPZV<1>, ZPZV<672>, ZPZV<5»; ); // NOLINT template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
04794
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
04795
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<673, 5> { using ZPZ = aerobus::zpz<673>; using type =
04796
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<668»; }; // NOLINT
04797
              template<> struct ConwayPolynomial<673, 6> { using ZPZ = aerobus::zpz<673>; using type =
        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 = DIVV<ZPZV<1 | PZZV | P
04798
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<668»; };
               template<> struct ConwayPolynomial<673, 8> { using ZPZ = aerobus::zpz<673>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<669>, ZPZV<587>, ZPZV<302>, ZPZV<5»; }; //
              template<> struct ConwayPolynomial<673, 9> { using ZPZ = aerobus::zpz<673>; using type =
04800
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<347>, ZPZV<553>, ZPZV<668»;
         }; // NOLINT
               template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
         POLYV<ZPZV<1>, ZPZV<675»; }; // NOLINT
04802
               template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
        POLYV<ZPZV<1>, ZPZV<672>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<677, 3> { using ZPZ = aerobus::zpz<677>; using type =
04803
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<675»; }; // NOLINT template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2»; }; // NOLINT
04805
               template<> struct ConwayPolynomial<677, 5> { using ZPZ = aerobus::zpz<677>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<675»; }; // NOLINT
        template<> struct ConwayPolynomial<677, 6> { using ZPZ = aerobus::zpz<677>; using type = POLYV<ZPZVX1>, ZPZV<0>, ZPZV<1>, ZPZV<446>, ZPZV<632>, ZPZV<50>, ZPZV<2x; }; // NOLINT template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type =
04806
04807
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<675»; };
04808
              template<> struct ConwayPolynomial<677, 8> { using ZPZ = aerobus::zpz<677>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<619>, ZPZV<619>, ZPZV<152>, ZPZV<2»; }; //
         NOLINT
        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<504>, ZPZV<504>, ZPZV<404>, ZPZV<675»;
04809
04810
               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 =
04811
         POLYV<ZPZV<1>. ZPZV<682>. ZPZV<5»: 1: // NOLINT
04812
               template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<678»; }; // NOLINT template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type =
04813
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<683, 5> { using ZPZ = aerobus::zpz<683>; using type =
04814
        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 =
04815
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<644>, ZPZV<109>, ZPZV<434>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<683, 7> { using ZPZ = aerobus::zpz<683>; using type =
04816
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<678»; };
04817
              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»; };
         NOLINT
04818
               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<0>, ZPZV<0>, ZPZV<85>, ZPZV<8444>, ZPZV<678»;
         }; // NOLINT
04819
               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 =
04820
        POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; }; // NOLINT
               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 =
04822
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<691, 5> { using ZPZ = aerobus::zpz<691>; using type =
04823
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; }; // NOLINT
04824
               template<> struct ConwayPolynomial<691, 6> { using ZPZ = aerobus::zpz<691>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<408>, ZPZV<262>, ZPZV<3»; }; // NOLINT
04825
              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<0>, ZPZV<4>, ZPZV<4>, ZPZV<48, ZPZV<688»; }; // NOLINT template<> struct ConwayPolynomial<691, 8> { using ZPZ = aerobus::zpz<691>; using type =
04826
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<356>, ZPZV<425>, ZPZV<321>, ZPZV<32); //
04827
              template<> struct ConwayPolynomial<691, 9> { using ZPZ = aerobus::zpz<691>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<55>, ZPZV<556>, ZPZV<443>, ZPZV<688»;
         }; // NOLINT
04828
               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 =
        POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; }; // NOLINT
04830
              template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; }; // NOLINT template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2»; }; // NOLINT
04831
```

```
04832
                template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699»; }; // NOLINT
04833
               template<> struct ConwayPolynomial<701, 6> { using ZPZ = aerobus::zpz<701>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<571>, ZPZV<327>, ZPZV<285>, ZPZV<28; }; // NOLINT template<> struct ConwayPolynomial<701, 7> { using ZPZ = aerobus::zpz<701>; using type
04834
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<699»; };
                                                                                                                                              // NOLINT
               template<> struct ConwayPolynomial<701, 8> { using ZPZ = aerobus::zpz<701>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<619>, ZPZV<206>, ZPZV<593>, ZPZV<593>, ZPZV<2»; }; //
         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<45>, ZPZV<45>, ZPZV<459>, ZPZV<373>, ZPZV<699»;</pre>
04836
         }; // NOLINT
04837
                template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
         POLYV<ZPZV<1>, ZPZV<707»; }; // NOLINT
04838
               template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
         POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
04839
         POLYVCZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<707»; }; // NOLINT template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; }; // NOLINT
                template<> struct ConwayPolynomial<709, 5> { using ZPZ = aerobus::zpz<709>; using type =
         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 =
04842
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<669>, ZPZV<514>, ZPZV<295>, ZPZV<295; ; // NOLINT template<> struct ConwayPolynomial<709, 7> { using ZPZ = aerobus::zpz<709>; using type
04843
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<707»; }; //
04844
               template<> struct ConwayPolynomial<709, 8> { using ZPZ = aerobus::zpz<709>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<689>, ZPZV<233>, ZPZV<79>, ZPZV<2»; };
         NOLINT
04845
               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<171>, ZPZV<707»;
         }; // NOLINT
               template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
04846
         POLYV<ZPZV<1>, ZPZV<708»; }; // NOLINT
                template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
04847
         POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; }; // NOLINT
         template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
04848
04849
                template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<5>, ZPZV<602>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<719, 5> { using ZPZ = aerobus::zpz<719>; using type =
04850
         template<> struct ConwayPolynomial</19, 5> { using ZPZ = aerobus::zpz</19>; using type = PoLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT 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 =
04851
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<708»; };
04853
              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»; }; //
         NOLINT
               template<> struct ConwayPolynomial<719, 9> { using ZPZ = aerobus::zpz<719>; using type =
04854
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<288>, ZPZV<560>, ZPZV<708»;
         }; // NOLINT
04855
                template<> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
         POLYV<ZPZV<1>, ZPZV<722»; }; // NOLINT
               template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
04856
         POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; }; // NOLINT
                template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<7>, ZPZY<7>, ZPZY<7>, ZPZY<722»; }; // NOLINT template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
04858
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5»; }; // NOLINT
        template<> struct ConwayPolynomial</27, 5> { using ZPZ = aerobus::zpz<727>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<722»; }; // NOLINT
04859
04860
                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
04861
               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<17>, ZPZV<722»; }; // NOLINT template<> struct ConwayPolynomial<727, 8> { using ZPZ = aerobus::zpz<727>; using type =
04862
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<639>, ZPZV<671>, ZPZV<368>, ZPZV<36»; }; //
         NOLINT
04863
                template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<573>, ZPZV<502>, ZPZV<722»;
         }; // NOLINT
04864
               template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
         POLYV<ZPZV<1>, ZPZV<727»; }; // NOLINT
               template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
04865
         POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; }; // NOLINT
               template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
04866
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727»; }; // NOLINT template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
04867
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<733, 5> { using ZPZ = aerobus::zpz<733>; using type =
04868
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<727»; }; // NOLINT
                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
04870
              template<> struct ConwayPolynomial<733, 7> { using ZPZ = aerobus::zpz<733>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04871
```

9.3 aerobus.h 169

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<532>, ZPZV<610>, ZPZV<142>, ZPZV<6*; }; //
04872
           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<0>, ZPZV<2>, ZPZV<2337>, ZPZV<6>, ZPZV<727»; };
       // NOLINT
04873
            template<> struct ConwavPolynomial<739. 1> { using ZPZ = aerobus::zpz<739>; using type =
       POLYV<ZPZV<1>, ZPZV<736»; }; // NOLINT
            template<> struct ConwayPolynomial<739, 2> { using ZPZ = aerobus::zpz<739>; using type =
04874
       POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; }; // NOLINT
04875
           template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; }; // NOLINT

template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<3»; }; // NOLINT
04876
            template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<736»; }; // NOLINT
04878
            template<> struct ConwayPolynomial<739, 6> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZVV1>, ZPZV<0>, ZPZV<0>, ZPZV<422>, ZPZV<447>, ZPZV<625>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<739, 7> { using ZPZ = aerobus::zpz<739>; using type
04879
       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 =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<401>, ZPZV<169>, ZPZV<25>, ZPZV<3»; };
       NOLINT
04881
           template<> struct ConwayPolynomial<739, 9> { using ZPZ = aerobus::zpz<739>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<616>, ZPZV<81>, ZPZV<736»;
       }; // NOLINT
04882
            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 =
04883
       POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<743, 3> { using ZPZ = aerobus::zpz<743>; using type =
04884
       POLYVCZPZV<1>, ZPZV<3>, ZPZV<38»; }; // NOLINT template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
04885
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<5»; }; // NOLINT
04886
           template<> struct ConwayPolynomial<743, 5> { using ZPZ = aerobus::zpz<743>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT template<> struct ConwayPolynomial<743, 6> { using ZPZ = aerobus::zpz<743>; using type =
04887
       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<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<738»; };
           template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>; }; //
       NOLINT
           template<> struct ConwayPolynomial<743, 9> { using ZPZ = aerobus::zpz<743>; using type =
04890
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32, ZPZV<327>, ZPZV<6762, ZPZV<738»;
04891
            template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
       POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
           template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
04892
       POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
04893
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; }; // NOLINT template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
04894
       04895
           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 =
04896
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<298>, ZPZV<633>, ZPZV<539>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<751,
                                                           7> { using ZPZ = aerobus::zpz<751>; using type
04897
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<75, ZPZV<75; // NOLINT template<> struct ConwayPolynomial<751, 8> { using ZPZ = aerobus::zpz<751>; using type =
04898
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<741>, ZPZV<243>, ZPZV<672>, ZPZV<672>, ZPZV<3»; }; //
       NOLINT
04899
            template<> struct ConwayPolynomial<751, 9> { using ZPZ = aerobus::zpz<751>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<703>, ZPZV<489>, ZPZV<7489;
       }; // NOLINT
04900
           template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
       POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
           template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
04901
       POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
04902
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
04903
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<757, 5> { using ZPZ = aerobus::zpz<757>; using type =
04904
       POLYY<ZPZY<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<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<753>, ZPZV<739>, ZPZV<745>, ZPZV<2»; }; // NOLINT
04906
           template<> struct ConwayPolynomial<757, 7> { using ZPZ = aerobus::zpz<757>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<755»; }; // NOLINT 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»; }; //
04907
       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<68>, ZPZV<688>, ZPZV<608>, ZPZV<755»;
       }; // NOLINT
            template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
04909
       POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
```

170 File Documentation

```
04910
           template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; }; // NOLINT
04911
           template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<6»; }; // NOLINT
04912
           template<> struct ConwayPolynomial<761, 5> { using ZPZ = aerobus::zpz<761>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT
04914
           template<> struct ConwayPolynomial<761, 6> { using ZPZ = aerobus::zpz<761>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<634>, ZPZV<597>, ZPZV<155>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<761, 7> { using ZPZ = aerobus::zpz<761>; using type =
04915
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<761, 8> { using ZPZ = aerobus::zpz<761>; using type =
04916
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<603>, ZPZV<144>, ZPZV<540>, ZPZV<54»; }; //
       NOLINT
04917
           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
            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 =
04919
      POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
04920
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; }; // NOLINT template<> struct ConwayPolynomial<769, 4> { using ZPZ = aerobus::zpz<769>; using type =
04921
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; }; // NOLINT
04922
           template<> struct ConwayPolynomial<769, 5> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<758»; }; // NOLINT
04923
           template<> struct ConwayPolynomial<769, 6> { using ZPZ = aerobus::zpz<769>; using type =
      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
04924
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<8>, ZPZV<758»; };
           template<> struct ConwayPolynomial<769, 8> { using ZPZ = aerobus::zpz<769>; using type
04925
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<560>, ZPZV<574>, ZPZV<632>, ZPZV<11»; }; //
       NOLINT
04926
           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<0>, ZPZV<623>, ZPZV<751>, ZPZV<758»;
04927
            template<> struct ConwayPolynomial<773, 1> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<771»; }; // NOLINT
04928
           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 =
04929
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<771»; }; // NOLINT template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2»; }; // NOLINT
04931
           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 =
04932
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<91>, ZPZV<3>, ZPZV<581>, ZPZV<2»; }; // NOLINT
04933
           template<> struct ConwayPolynomial<773, 7> { using ZPZ = aerobus::zpz<773>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9, ZPZV<771»; }; // NoLII template<> struct ConwayPolynomial<773, 8> { using ZPZ = aerobus::zpz<773>; using type :
04934
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<484>, ZPZV<94>, ZPZV<693>, ZPZV<2»; }; //
       NOLINT
      template<> struct ConwayPolynomial<773, 9> { using ZPZ = aerobus::zpz<773>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<216>, ZPZV<216>, ZPZV<771»;
04935
       }; // NOLINT
           template<> struct ConwayPolynomial<787, 1> { using ZPZ = aerobus::zpz<787>; using type =
04936
      POLYV<ZPZV<1>, ZPZV<785»; }; // NOLINT
           template<> struct ConwayPolynomial<787, 2> { using ZPZ = aerobus::zpz<787>; using type =
04937
      POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
04938
           template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<785»; }; // NOLINT template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type =
04939
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
04940
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<785»; }; // NOLINT
           template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
04941
      POLYV<ZPZV<1>, 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 =
04942
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<785»; }; // NOLINT
04943
           template<> struct ConwayPolynomial<787, 8> { using ZPZ = aerobus::zpz<787>; using type =
       POLYV<2PZV<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<5>, ZPZV<480>, ZPZV<573>, ZPZV<785»;
       }; // NOLINT
04945
           template<> struct ConwayPolynomial<797, 1> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<795»; }; // NOLINT
           template<> struct ConwayPolynomial<797, 2> { using ZPZ = aerobus::zpz<797>; using type =
04946
      POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<795»; };
                                                             // NOLINT
          template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type =
04948
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<177, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<797, 5> { using ZPZ = aerobus::zpz<797>; using type =
```

9.3 aerobus.h

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<795»; };
           template<> struct ConwayPolynomial<797, 6> { using ZPZ = aerobus::zpz<797>; using type = 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 =
04951
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type =
04952
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2»; }; //
04953
                  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
04954
                   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 =
           POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
04956
                  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 =
04957
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; };
                                                                                                                        // NOLINT
                   template<> struct ConwayPolynomial<809, 5> { using ZPZ = aerobus::zpz<809>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; }; // NOLINT
04959
                   template<> struct ConwayPolynomial<809, 6> { using ZPZ = aerobus::zpz<809>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<56>, ZPZV<75>, ZPZV<43>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<809, 7> { using ZPZ = aerobus::zpz<809>; using type
04960
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; };
                   template<> struct ConwayPolynomial<809, 8> { using ZPZ = aerobus::zpz<809>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<593>, ZPZV<745>, ZPZV<673>, ZPZV
           NOLINT
           template<> struct ConwayPolynomial<809, 9> { using ZPZ = aerobus::zpz<809>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<341>, ZPZV<341>, ZPZV<341>, ZPZV<727>, ZPZV<806»;</pre>
04962
           }; // NOLINT
04963
                    template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
           POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
04964
                  template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
           POLYV<ZPZV<1>, ZPZV<806>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<811, 3> { using ZPZ = aerobus::zpz<811>; using type =
04965
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<608»; }; // 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
                  template<> struct ConwayPolynomial<811, 5> { using ZPZ = aerobus::zpz<811>; using type =
04967
           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 =
04968
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<780>, ZPZV<755>, ZPZV<307>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<811, 7> { using ZPZ = aerobus::zpz<811>; using type
04969
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<808»; }; // NOLINT
04970
                   template<> struct ConwayPolynomial<811, 8> { using ZPZ = aerobus::zpz<811>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<525>, ZPZV<3»; }; //
           NOLINT
                  template<> struct ConwayPolynomial<811, 9> { using ZPZ = aerobus::zpz<811>; using type =
04971
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<382>, ZPZV<200>, ZPZV<808»;
           }; // NOLINT template<> struct ConwayPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
04972
           POLYV<ZPZV<1>, ZPZV<819»; }; // NOLINT template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
04973
           POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT
                   template<> struct ConwayPolynomial<821, 3> { using ZPZ = aerobus::zpz<821>; using type =
04974
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<819»; }; // NOLINT
                  template<> struct ConwayPolynomial<821, 4> { using ZPZ = aerobus::zpz<821>; using type =
04975
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<15>, ZPZY<662>, ZPZY<2*; }; // NOLINT template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
04976
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8), ZPZV<8), ZPZV<819»; }; // NOLINT template<> struct ConwayPolynomial<821, 6> { using ZPZ = aerobus::zpz<821>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<160>, ZPZV<130>, ZPZV<803>, ZPZV<2»; }; // NOLINT
04977
                    template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10
04979
                  template<> struct ConwayPolynomial<821, 8> { using ZPZ = aerobus::zpz<821>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<626>, ZPZV<556>, ZPZV<589>, ZPZV<2*; }; //
           NOLINT
                  template<> struct ConwayPolynomial<821, 9> { using ZPZ = aerobus::zpz<821>; using type =
04980
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<650>, ZPZV<650>, ZPZV<557>, ZPZV<819»;
           }; // NOLINT
04981
                  template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
           POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
                   template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
04982
           POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
04984
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
04985
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
04986
                   template<> struct ConwayPolynomial<823, 6> { using ZPZ = aerobus::zpz<823>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<82>, ZPZV<616>, ZPZV<744>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type =
04987
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<820»; }; // NOLINT 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»; }; //
04988
```

172 File Documentation

```
NOLINT
        template<> struct ConwayPolynomial<823, 9> { using ZPZ = aerobus::zpz<823>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<740>, ZPZV<609>, ZPZV<820»;
         }; // NOLINT
04990
               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 =
         POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT
04992
              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 =
04993
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<827, 5> { using ZPZ = aerobus::zpz<827>; using type =
04994
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<825»; }; // NOLINT
04995
              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 template<> struct ConwayPolynomial<827, 7> { using ZPZ = aerobus::zpz<827>; using type =
04996
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<812>, ZPZV<79>, ZPZV<32>, ZPZV<32>; };
04998
               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<0>, ZPZV<177>, ZPZV<372>, ZPZV<825»;
         }; // NOLINT
04999
               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 =
05000
        POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; }; // NOLINT
05001
               template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<827»; }; // NOLINT template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
05002
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<604>, ZPZV<2»; };
                                                                                               // NOLINT
               template<> struct ConwayPolynomial<829, 5> { using ZPZ = aerobus::zpz<829>; using type =
05003
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<827»; }; // NOLINT
05004
               template<> struct ConwayPolynomial<829, 6> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<3476>, ZPZV<817>, ZPZV<829; }; // NOLINT template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type
05005
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<827»; }; //
               template<> struct ConwayPolynomial<829, 8> { using ZPZ = aerobus::zpz<829>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<468>, ZPZV<241>, ZPZV<138>, ZPZV<2»; }; //
         NOLINT
        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<62>, ZPZV<621>, ZPZV<621>, ZPZV<552>, ZPZV<827»;</pre>
05007
         }; // NOLINT
               template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<828»; }; // NOLINT
05009
              template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; }; // NOLINT
               template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
05010
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<4>, ZPZY<828»; }; // NOLINT template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
05011
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11»; }; // NOLINT
05012
               template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<8288; }; // NOLINT template<> struct ConwayPolynomial<839, 6> { using ZPZ = aerobus::zpz<839>; using type =
05013
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<370>, ZPZV<23>, ZPZV<23>, ZPZV<23>, ZPZV<23>; // NOLINT template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<828»; };
              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
        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<349>, ZPZV<206>, ZPZV<828*;
05016
         }; // NOLINT
05017
              template<> struct ConwayPolynomial<853, 1> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<ZPZV<1>, ZPZV<851»; }; // NOLINT
05018
               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 =
05019
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
05020
        template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
05021
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<851»; }; // NOLINT
               template<> struct ConwayPolynomial<853, 6> { using ZPZ = aerobus::zpz<853>; using type =
05022
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<276>, ZPZV<194>, ZPZV<512>, ZPZV<2»; }; // NOLINI
               template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type
05023
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 8> { using ZPZ = aerobus::zpz<853>; using type =
05024
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<544>, ZPZV<846>, ZPZV<118>, ZPZV<118>, ZPZV<2»; }; //
         NOLINT
05025
               template<> struct ConwayPolynomial<853, 9> { using ZPZ = aerobus::zpz<853>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<677>, ZPZV<621>, ZPZV<851»;
         }; // NOLINT
05026
              POLYV<ZPZV<1>, ZPZV<854»; }; // NOLINT
              template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
```

9.3 aerobus.h 173

```
POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; };
                          template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; }; // NOLINT template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
05029
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<528>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<857, 5> { using ZPZ = aerobus::zpz<857>; using type =
05030
               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 =
05031
                \verb"POLYV<2PZV<1>, \verb"ZPZV<0>, \verb"ZPZV<1>, \verb"ZPZV<32>, \verb"ZPZV<824>, \verb"ZPZV<65>, \verb"ZPZV<3*"; \verb"]; $ // \verb"NOLINT" | NOLINT" | NOL
05032
                        template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::zpz<857>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<854»; }; // NOLINT template<> struct ConwayPolynomial<857, 8> { using ZPZ = aerobus::zpz<857>; using type =
05033
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<611>, ZPZV<552>, ZPZV<494>, ZPZV<3»; }; //
05034
                        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
05035
                          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 =
               POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT
05037
                         template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857»; }; // NOLINT
template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2»; }; // NOLINT
05038
                          template<> struct ConwayPolynomial<859, 5> { using ZPZ = aerobus::zpz<859>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<857»; }; // NOLINT
05040
                         template<> struct ConwayPolynomial<859, 6> { using ZPZ = aerobus::zpz<859>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<419>, ZPZV<646>, ZPZV<566>, ZPZV<2»; }; // NOLINT
05041
                        template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpz<859>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05042
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<52>, ZPZV<446>, ZPZV<672>, ZPZV<672>; }; //
               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»;</pre>
05043
               }; // NOLINT
                          template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
               POLYV<ZPZV<1>, ZPZV<858»; }; // NOLINT
                         template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
05045
               POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; }; // NOLINT
                         template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
05046
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<858»; }; // NOLINT template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
05047
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5»; }; // NOLINT
05048
                         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
05049
                        template<> struct ConwayPolynomial<863, 6> { using ZPZ = aerobus::zpz<863>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<330>, ZPZV<62>, ZPZV<300>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<863, 7> { using ZPZ = aerobus::zpz<863>; using type =
05050
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<858»; };
                         template<> struct ConwayPolynomial<863, 8> { using ZPZ = aerobus::zpz<863>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<576>, ZPZV<576>, ZPZV<849>, ZPZV<5%; }; //
05052
                        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»; };
05053
                        template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
               POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
05054
                         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 =
05055
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875»; }; // NOLINT
              template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; }; // NOLINT
05057
                        template<> struct ConwayPolynomial<877, 5> { using ZPZ = aerobus::zpz<877>; using type =
                \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<6>, \verb"ZPZV<6>, \verb"ZPZV<875"; \verb"// NOLINT" | NOLINT | N
              template<> struct ConwayPolynomial<877, 6> { using ZPZ = aerobus::zpz<877>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<400>, ZPZV<855>, ZPZV<855>, ZPZV<2»; }; // NOLINT
05058
                          template<> struct ConwayPolynomial<877,
                                                                                                                              7> { using ZPZ = aerobus::zpz<87</pre>
                                                                                                                                                                                                                  .
7>; using type
               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3
05060
                       template<> struct ConwayPolynomial<877, 8> { using ZPZ = aerobus::zpz<877>; using type =
               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 =
05061
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<770>, ZPZV<770>, ZPZV<278>, ZPZV<875»;
05062
                         template<> struct ConwayPolynomial<881, 1> { using ZPZ = aerobus::zpz<881>; using type =
               POLYV<ZPZV<1>, ZPZV<878»; }; // NOLINT
                         template<> struct ConwayPolynomial<881, 2> { using ZPZ = aerobus::zpz<881>; using type =
05063
               POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT
05064
                         template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<878»; }; // NOLINT template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
05065
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<447>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<881, 5> { using ZPZ = aerobus::zpz<881>; using type =
05066
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<878»; }; // NOLINT
```

174 File Documentation

```
05067
                 template<> struct ConwayPolynomial<881, 6> { using ZPZ = aerobus::zpz<881>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<419>, ZPZV<213>, ZPZV<23»; }; // NOLINT template<> struct ConwayPolynomial<881, 7> { using ZPZ = aerobus::zpz<881>; using type =
05068
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<878»; }; // NOLINT template<> struct ConwayPolynomial<881, 8> { using ZPZ = aerobus::zpz<881>; using type =
05069
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<635>, ZPZV<490>, ZPZV<561>, ZPZV<561>, ZPZV<3»; }; //
                 template<> struct ConwayPolynomial<881, 9> { using ZPZ = aerobus::zpz<881>; using type
05070
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<587>, ZPZV<510>, ZPZV<878»;
          }; // NOLINT
05071
                template<> struct ConwayPolynomial<883, 1> { using ZPZ = aerobus::zpz<883>; using type =
         POLYV<ZPZV<1>, ZPZV<881»; }; // NOLINT
                template<> struct ConwayPolynomial<883, 2> { using ZPZ = aerobus::zpz<883>; using type =
05072
          POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; }; // NOLINT
05073
                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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<883, 5> { using ZPZ = aerobus::zpz<883>; using type =
05074
         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 =
05076
         POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<879>, ZPZV<865>, ZPZV<871>, ZPZV<2»; }; // NOLINT
05077
                template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05078
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<740>, ZPZV<762>, ZPZV<768>, ZPZV<20»; }; //
         template<> struct ConwayPolynomial<883, 9> { using ZPZ = aerobus::zpz<883>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<360>, ZPZV<360>, ZPZV<881»;</pre>
05079
          }; // NOLINT
                 template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
05080
         POLYV<ZPZV<1>, ZPZV<882»; }; // NOLINT
                template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
05081
          POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; }; // NOLINT
05082
                 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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5»; }; // NOLINT
05083
05084
                 template<> struct ConwayPolynomial<887, 5> { using ZPZ = aerobus::zpz<887>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<882»; }; // NOLINT
         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 =
05085
05086
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<82»; };
                template<> struct ConwayPolynomial<887, 8> { using ZPZ = aerobus::zpz<887>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<781>, ZPZV<381>, ZPZV<706>, ZPZV<70s, }; //
         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»;</pre>
05088
          }; // NOLINT
05089
                 template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
          POLYV<ZPZV<1>, ZPZV<905»; }; // NOLINT
05090
                template<> struct ConwayPolynomial<907, 2> { using ZPZ = aerobus::zpz<907>; using type =
         POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<907, 3> { using ZPZ = aerobus::zpz<907>; using type =
05091
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<4>, ZPZV<905»; }; // NOLINT template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
05093
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; }; // NOLINT
                template<> struct ConwayPolynomial<907, 6> { using ZPZ = aerobus::zpz<907>; using type =
05094
         POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<626, ZPZV<2665, ZPZV<2665, ZPZV<22*; }; // NOLINT template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type
05095
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<905»; }; //
05096
                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<2»; }; //
          NOLINT
05097
                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<1>, ZPZV<783>, ZPZV<57>, ZPZV<905»;
          }; // NOLINT
                template<> struct ConwayPolynomial<911, 1> { using ZPZ = aerobus::zpz<911>; using type =
05098
         POLYV<ZPZV<1>, ZPZV<894»; }; // NOLINT
                template<> struct ConwayPolynomial<911, 2> { using ZPZ = aerobus::zpz<911>; using type =
05099
         POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; }; // NOLINT
         template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; }; // NOLINT
05100
                template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
05101
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
05102
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2894»; }; // NOLINT template<> struct ConwayPolynomial<911, 6> { using ZPZ = aerobus::zpz<911>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<683>, ZPZV<19>, ZPZV<17»; }; // NOLINT
05103
                 template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<894»; };
05105
               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»; }; //
          NOLTNT
```

9.3 aerobus.h 175

```
template<> struct ConwayPolynomial<911, 9> { using ZPZ = aerobus::zpz<911>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<67, ZPZV<679>, ZPZV<6166, ZPZV<894»;
         }; // NOLINT
05107
              template<> struct ConwayPolynomial<919, 1> { using ZPZ = aerobus::zpz<919>; using type =
         POLYV<ZPZV<1>, ZPZV<912»; }; // NOLINT
               template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
05108
         POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; }; // NOLINT
05109
               template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912»; }; // NOLINT template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
0.5110
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<919, 5> { using ZPZ = aerobus::zpz<919>; using type =
05111
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<912»; }; // NOLINT
               template<> struct ConwayPolynomial<919, 6> { using ZPZ = aerobus::zpz<919>; using type =
05112
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<312>, ZPZV<817>, ZPZV<113>, ZPZV<7»; };
05113
              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<0>, ZPZV<9>, ZPZV<912»; }; // NOLINT template<> struct ConwayPolynomial<919, 8> { using ZPZ = aerobus::zpz<919>; using type =
05114
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<202>, ZPZV<504>, ZPZV<504>, ZPZV<5»; }; //
         template<> struct ConwayPolynomial<919, 9> { using ZPZ = aerobus::zpz<919>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<410>, ZPZV<623>, ZPZV<912»;
05115
         }; // NOLINT
               template<> struct ConwayPolynomial<929, 1> { using ZPZ = aerobus::zpz<929>; using type =
0.5116
         POLYV<ZPZV<1>, ZPZV<926»; }; // NOLINT
               template<> struct ConwayPolynomial<929, 2> { using ZPZ = aerobus::zpz<929>; using type =
         POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3»; }; // NOLINT
05118
               template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<926»; }; // NOLINT template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type =
05119
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<38; }; // NOLINT template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
05120
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<926»; }; // NOLINT
              template<> struct ConwayPolynomial<929, 6> { using ZPZ = aerobus::zpz<929>; using type =
0.5121
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<805>, ZPZV<92>, ZPZV<86>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type =
05122
         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»; };
05124
              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<0>, ZPZV<481>, ZPZV<199>, ZPZV<926»;
         }; // NOLINT
05125
               template<> struct ConwayPolynomial<937, 1> { using ZPZ = aerobus::zpz<937>; using type =
         POLYV<ZPZV<1>, ZPZV<932»; }; // NOLINT
05126
               template<> struct ConwayPolynomial<937, 2> { using ZPZ = aerobus::zpz<937>; using type =
         POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5»; }; // NOLINT
0.512.7
              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 =
05128
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<23>, ZPZV<585>, ZPZV<5»; };
                                                                                                 // NOLINT
               template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<932»; }; // NOLINT
        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 =
05130
05131
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<932»; }; //
              template<> struct ConwayPolynomial<937, 8> { using ZPZ = aerobus::zpz<937>; using type =
05132
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<265, ZPZV<53>, ZPZV<53>, ZPZV<59, ZPZV<50, Z
         NOLINT
05133
              template<> struct ConwayPolynomial<937, 9> { using ZPZ = aerobus::zpz<937>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<533>, ZPZV<483>, ZPZV<932»;
         }; // NOLINT
               template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
         POLYV<ZPZV<1>, ZPZV<939»; }; // NOLINT
05135
              template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
         POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
05136
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<939»; }; // NOLINT
               template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type =
05137
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<941, 5> { using ZPZ = aerobus::zpz<941>; using type =
05138
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 6> { using ZPZ = aerobus::zpz<941>; using type =
05139
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<459, ZPZV<538>, ZPZV<238>, ZPZV<29; }; // NOLINT template<> struct ConwayPolynomial<941, 7> { using ZPZ = aerobus::zpz<941>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<3939»; };
05141
              template<> struct ConwayPolynomial<941, 8> { using ZPZ = aerobus::zpz<941>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<675>, ZPZV<675>, ZPZV<590>, ZPZV<2»; }; //
         NOLINT
               template<> struct ConwayPolynomial<941, 9> { using ZPZ = aerobus::zpz<941>; using type =
05142
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708>, ZPZV<197>, ZPZV<939%;
         }; // NOLINT
05143
               template<> struct ConwayPolynomial<947, 1> { using ZPZ = aerobus::zpz<947>; using type =
        POLYV<ZPZV<1>, ZPZV<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
0.5144
         POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2»; }; // NOLINT
```

176 File Documentation

```
05145
               template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 4> { using ZPZ = aerobus::zpz<947>; using type =
05146
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<947, 5> { using ZPZ = aerobus::zpz<947>; using type =
0.5147
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<945»; }; // NOLINT
               template<> struct ConwayPolynomial<947, 6> { using ZPZ = aerobus::zpz<947>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<880>, ZPZV<787>, ZPZV<95>, ZPZV<2»; }; // NOLINI
              template<> struct ConwayPolynomial<947, 7> { using ZPZ = aerobus::zpz<947>; using type =
05149
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 8> { using ZPZ = aerobus::zpz<947>; using type =
05150
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<845>, ZPZV<597>, ZPZV<581>, ZPZV<2*; }; //
               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<1>, ZPZV<26>, ZPZV<269>, ZPZV<808>, ZPZV<945»;
         }; // NOLINT
05152
               template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
         POLYV<ZPZV<1>, ZPZV<950»; }; // NOLINT
               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 =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950»; }; // NOLINT template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
05155
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<865>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<953, 5> { using ZPZ = aerobus::zpz<953>; using type =
05156
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<950»; }; // NOLINT
               template<> struct ConwayPolynomial<953, 6> { using ZPZ = aerobus::zpz<953>; using type =
05157
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<507>, ZPZV<829>, ZPZV<730>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<953, 7> { using ZPZ = aerobus::zpz<953>; using type
05158
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5
              template<> struct ConwayPolynomial<953, 8> { using ZPZ = aerobus::zpz<953>; using type =
05159
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<579>, ZPZV<658>, ZPZV<108>, ZPZV<3»; }; //
              template<> struct ConwayPolynomial<953, 9> { using ZPZ = aerobus::zpz<953>; using type =
05160
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<819>, ZPZV<316>, ZPZV<950»;
         }; // NOLINT
         template<> struct ConwayPolynomial<967, 1> { using ZPZ = aerobus::zpz<967>; using type =
POLYV<ZPZV<1>, ZPZV<962»; }; // NOLINT</pre>
05161
                template<> struct ConwayPolynomial<967, 2> { using ZPZ = aerobus::zpz<967>; using type =
         POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5»; }; // NOLINT
05163
              template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; }; // NOLINT
05164
               template<> struct ConwayPolynomial<967, 5> { using ZPZ = aerobus::zpz<967>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<962»; }; // NOLINT
05166
              template<> struct ConwayPolynomial<967, 6> { using ZPZ = aerobus::zpz<967>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<948>, ZPZV<831>, ZPZV<5»; }; // NOLINT
05167
               template<> struct ConwayPolynomial<967, 7> { using ZPZ = aerobus::zpz<967>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<962*; }; // NOLINT
               template<> struct ConwayPolynomial<967, 8> { using ZPZ = aerobus::zpz<967>; using type
05168
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<840>, ZPZV<502>, ZPZV<136>, ZPZV<5»; }; //
         NOLINT
         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<51>, ZPZV<512>, ZPZV<783>, ZPZV<962»;</pre>
05169
         }; // NOLINT
                template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
         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
              template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
05172
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<965»; }; // NOLINT template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
05173
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<527>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
05174
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<965»; }; // NOLINT
05175
               template<> struct ConwayPolynomial<971, 6> { using ZPZ = aerobus::zpz<971>; using type =
        POLYV-ZPZV-1>, ZPZV-0>, ZPZV-1>, ZPZV-970>, ZPZV-718>, ZPZV-718>, ZPZV-6>; }; // NOLINT template<> struct ConwayPolynomial<971, 7> { using ZPZ = aerobus::zpz<971>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<965»; }; //
               template<> struct ConwayPolynomial<971, 8> { using ZPZ = aerobus::zpz<971>; using type =
05177
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<281>, ZPZV<206>, ZPZV<6»; }; //
         NOLINT
05178
               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<0>, ZPZV<0>, ZPZV<805>, ZPZV<473>, ZPZV<965»;
         }; // NOLINT
05179
               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 =
05180
         POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<977, 3> { using ZPZ = aerobus::zpz<977>; using type =
05181
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<974»; }; // NOLINT
        template<> struct ConwayPolynomial
+ 8 | struct ConwayPolynomial
+ 8 | struct ConwayPolynomial
+ 90LYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<800>, ZPZV<3»; }; // NOLINT
05183
              template<> struct ConwayPolynomial<977, 5> { using ZPZ = aerobus::zpz<977>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<974»; }; // NOLINT
              template<> struct ConwayPolynomial<977, 6> { using ZPZ = aerobus::zpz<977>; using type =
05184
```

9.3 aerobus.h 177

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<729>, ZPZV<830>, ZPZV<753>, ZPZV<3»; };
           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<7+, ZPZV<974»; }; // NOLINT
          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»; }; //
      NOT.TNT
           template<> struct ConwayPolynomial<977, 9> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<740>, ZPZV<974»;
      }; // NOLINT
05188
           template<> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<978»; }; // NOLINT
           template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
05189
      POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978»; }; // NOLINT template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
0.5191
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<567>, ZPZV<567>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
05192
      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 template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type =
05194
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<978»; }; // NOLINT template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
05195
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<5w; }; //
          template<> struct ConwayPolynomial<983, 9> { using ZPZ = aerobus::zpz<983>; using type =
05196
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<858>, ZPZV<87>, ZPZV<978»;
      }; // NOLINT
05197
           template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<985»; }; // NOLINT
05198
           template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; }; // NOLINT
05199
          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 =
05200
      POLYVCZPZV<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
           template<> struct ConwayPolynomial<991, 6> { using ZPZ = aerobus::zpz<991>; using type =
05202
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<637>, ZPZV<855>, ZPZV<278>, ZPZV<6»; }; // NOLINT
05203
           template<> struct ConwayPolynomial<991, 7> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<985»; }; // NOLINT
           template<> struct ConwayPolynomial<991, 8> { using ZPZ = aerobus::zpz<991>; using type =
05204
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<941>, ZPZV<786>, ZPZV<234>, ZPZV<6»; }; //
      NOLINT
05205
          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<4222>, ZPZV<985»;
      }; // NOLINT
05206
           template<> struct ConwavPolynomial<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 =
      POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
05208
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990»; }; // NOLINT template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
05209
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7»; };
                                                                       // NOLINT
          template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
05210
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<990»; }; // NOLINT
05211
           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
05212
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<990»; }; // NOLINT
           template<> struct ConwayPolynomial<997, 8> { using ZPZ = aerobus::zpz<997>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<473>, ZPZV<4473 , ZPZV<241>, ZPZV<7»; }; //
      NOLINT
      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»;
05214
      }; // NOLINT
05215 #endif // DO_NOT_DOCUMENT
         // namespace aerobus
05216 }
05217 #endif // AEROBUS_CONWAY_IMPORTS
05218
05219 #endif // INC AEROBUS // NOLINT
```

178 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>

180 Examples

### 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

division operator yields v1 / v2 <i32::val<7>, i32::val<2>> -> i32::val<3> division operator yields v1 / v2 <i32::val<7>, i32::val<2>> -> i32::val<3>

v1	a value in i32
v2	a value in i32

10.11 i32::gcd\_t 181

### 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<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral\_constant<br/> < 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&lt;1&gt;&gt;</i32::val<1>

### 10.11 i32::gcd\_t

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

v1	a value in i32
v2	a value in i32

182 Examples

### 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&lt;2&gt;&gt;</i64::val<1>

10.19 i64::mod\_t 183

# 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&lt;2&gt;&gt;</i64::val<1>

# 10.17 i64::mul\_t

multiplication operator

multiplication operator

#### **Template Parameters**

v1	: an element of aerobus::i64::val
<i>v</i> 2	: an element of aerobus::i64::val <i64::val <1="">, i64::val &lt;2&gt;&gt;</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&lt;2&gt;&gt;</i64::val<1>

# 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 &lt;15&gt;&gt;</i64::val>	

184 Examples

### 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 &lt;1&gt;&gt;</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 > >	

### 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 &lt;2&gt;&gt;</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 &lt;2&gt;&gt;</i64::val>	

10.27 i64::pos\_v 185

### 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 &lt;2&gt;&gt;</i64::val>	١

### 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 &lt;15&gt;&gt;</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>>

186 Examples

### 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&lt;2&gt;&gt;, q32::val<i32::val<1>, i32::val&lt;3&gt;&gt;&gt;</i32::val<1></q32::val<i32::val<1>	Ī

### 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  $\pi$  as a continued fraction -> 3.14...

# 10.33 E\_fraction::val

approximation of e -> 2.718...

approximation of  $e \rightarrow 2.718...$ 

188 Examples

# 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
lt_v, 54	X, 61
mod_t, 52	zero, 61
mul_t, 52 one, 53	aerobus::polynomial< Ring >::val< coeffN >, 82 aN, 83
pos_t, 53	coeff_at_t, 83
pos_v, 54	degree, 84
sub_t, 53	enclosing type, 83
zero, 53	eval, 84
aerobus::i64::val $< x >$ , 74	is_zero_t, 83
enclosing_type, 75	is_zero_v, 84
eval, 75	strip, 83
get, 76	to_string, 84
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_ot_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)>>, $\frac{44}{4}$ type, $\frac{44}{4}$
make_index_sequence_reverse, 36 type_at_t, 36 aerobus::is_prime< n >, 54	$\label{eq:condition} $\inf(x, std::enable_if_t<(index==0)>>, \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 aN, 77 coeff_at_t, 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	index, std::enable_if_t<(index==0)>>, 44 type, 44 aerobus::polynomial< Ring >::val< coeffN, coeffs >, 76 aN, 77 coeff_at_t, 78 degree, 79
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 >,
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 >,  76  aN, 77  coeff_at_t, 78  degree, 79  enclosing_type, 78  eval, 78  is_zero_t, 78  is_zero_v, 79  strip, 78  to_string, 79
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	index, std::enable_if_t<(index==0)>>, 44  type, 44  aerobus::polynomial< Ring >::val< coeffN, coeffs >,  76  aN, 77  coeff_at_t, 78  degree, 79  enclosing_type, 78  eval, 78  is_zero_t, 78  is_zero_v, 79  strip, 78  to_string, 79  aerobus::Quotient< Ring, X >, 62
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	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	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_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 >,  76  aN, 77  coeff_at_t, 78  degree, 79  enclosing_type, 78  eval, 78  is_zero_t, 78  is_zero_v, 79  strip, 78  to_string, 79  aerobus::Quotient< Ring, X >, 62  add_t, 64  div_t, 64  eq_t, 64  eq_v, 66  inject_constant_t, 64
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	index, std::enable_if_t<(index==0)>>, 44  type, 44  aerobus::polynomial< Ring >::val< coeffN, coeffs >,  76  aN, 77  coeff_at_t, 78  degree, 79  enclosing_type, 78  eval, 78  is_zero_t, 78  is_zero_v, 79  strip, 78  to_string, 79  aerobus::Quotient< Ring, X >, 62  add_t, 64  div_t, 64  eq_t, 64  eq_v, 66  inject_constant_t, 64  inject_ring_t, 65
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 probabilist, 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::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 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 >,  76  aN, 77  coeff_at_t, 78  degree, 79  enclosing_type, 78  eval, 78  is_zero_t, 78  is_zero_v, 79  strip, 78  to_string, 79  aerobus::Quotient< Ring, X >, 62  add_t, 64  div_t, 64  eq_t, 64  eq_v, 66  inject_constant_t, 64  inject_ring_t, 65  is_euclidean_domain, 66  mod_t, 65  mul_t, 65  one, 65  pos_t, 65
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 >,  76  aN, 77  coeff_at_t, 78  degree, 79  enclosing_type, 78  eval, 78  is_zero_t, 78  is_zero_v, 79  strip, 78  to_string, 79  aerobus::Quotient< Ring, X >, 62  add_t, 64  div_t, 64  eq_t, 64  eq_v, 66  inject_constant_t, 64  inject_ring_t, 65  is_euclidean_domain, 66  mod_t, 65  mul_t, 65  one, 65  pos_t, 65  pos_v, 66
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 >,  76  aN, 77  coeff_at_t, 78  degree, 79  enclosing_type, 78  eval, 78  is_zero_t, 78  is_zero_v, 79  strip, 78  to_string, 79  aerobus::Quotient< Ring, X >, 62  add_t, 64  div_t, 64  eq_t, 64  eq_v, 66  inject_constant_t, 64  inject_ring_t, 65  is_euclidean_domain, 66  mod_t, 65  mul_t, 65  one, 65  pos_t, 65

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	,
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, 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
inject_constant_t, 87	aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN $>$ , 83
inner_type, 87	aerobus::polynomial< Ring >::val< coeffN, coeffs
is_euclidean_domain, 90	>, 78
is field, 90	combination_t
lt_t, 87	aerobus, 23
lt v, 90	combination_v
mod_t, 88	aerobus, 32
mul_t, 88	concat
one, 88	aerobus::type_list< Ts >, 69
pos_t, 88	aerobus::type_list<>, 71
pos_v, 90	COS
sub_t, 89	aerobus, 23
zero, 89	cosh
aerobus::zpz $<$ p $>$ ::val $<$ x $>$ , 80	aerobus, 24
enclosing_type, 81	
eval, 81	degree
get, 81	aerobus::polynomial< Ring >::val< coeffN >, 84
is_zero_t, 81	aerobus::polynomial< Ring >::val< coeffN, coeffs
to_string, 81	>, 79
v, 82	derive_t
aligned_malloc	aerobus::polynomial< Ring >, 57
aerobus, 31	div_t
alternate_t	aerobus::i32, 48
aerobus, 21	aerobus::i64, 51
alternate_v	aerobus::polynomial< Ring >, 57
aerobus, 31	aerobus::Quotient< Ring, X >, 64
aN	aerobus::zpz, 86
aerobus::polynomial< Ring >::val< coeffN >, 83	E fraction
, , , , , , , , , , , , , , , , , , , ,	E_fraction
	aerobus, 24

enclosing_type	aerobus::zpz $<$ p $>$ , 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 $<$ p $>$ , 87
aerobus::polynomial< Ring >::val< coeffN >, 84	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 Country
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 $<$ p $>$ , 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 >, 76	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	

>, 79  aerobus::i32, 49 aerobus::i64, 53  laguerre aerobus::known_polynomials, 39  legendre aerobus::known_polynomials, 40  length aerobus::type_list< Ts >, 70  aerobus::i32, 49 aerobus::polynomial< Ring >, 60 aerobus::Quotient< Ring, X >, 65 aerobus::zpz, 88 pos_v aerobus::i32, 50 aerobus::i64, 54
$\begin{array}{llllllllllllllllllllllllllllllllllll$
aerobus::known_polynomials, 39
$\begin{array}{llllllllllllllllllllllllllllllllllll$
aerobus::known_polynomials, 40 pos_v length aerobus::type_list< Ts >, 70 aerobus::i32, 50 aerobus::i64, 54
length aerobus::i32, 50 aerobus::i64, 54
aerobus::type_list< Ts >, 70 aerobus::i64, 54
•••••••••••••••••••••••••••••••••••••••
aerobus::type_list<>, 72 aerobus::polynomial< Ring >, 61
Inp1 aerobus::Quotient < Ring, X >, 66
aerobus, 26 aerobus:: $zpz  0$
It_t pow_t
aerobus::i32, 48 aerobus, 27
aerobus::i64, 52 pq64
aerobus::polynomial < Ring >, 58 aerobus, 27
aerobus::zpz, 87 probabilist
lt_v aerobus::known_polynomials, 40
aerobus::i64, 54 push_back
aerobus::zpz, 90 aerobus::type_list< Ts >, 69
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
— The state of the
aerobus, 26 aerobus, 28
aerobus, 26 aerobus, 28 mod_t
aerobus, 26 aerobus, 28 mod_t aerobus:i32, 48 README.md, 93
aerobus, 26 aerobus, 28  mod_t aerobus::i32, 48 README.md, 93 aerobus::i64, 52 remove
aerobus, 26  mod_t aerobus::i32, 48 aerobus::i64, 52 aerobus::polynomial < Ring >, 59  aerobus::type_list < Ts >, 70
aerobus, 26  mod_t  aerobus::i32, 48  aerobus::i64, 52  aerobus::polynomial < Ring >, 59  aerobus::Quotient < Ring, X >, 65  aerobus, 28  README.md, 93  remove  aerobus::type_list < Ts >, 70
aerobus, 26 aerobus, 28 aerobus, 28 mod_t README.md, 93 aerobus::i64, 52 remove aerobus::polynomial < Ring >, 59 aerobus::Quotient < Ring, X >, 65 aerobus::zpz , 88 simplify_t
aerobus, 26 aerobus, 28 aerobus, 28 mod_t aerobus::i32, 48 README.md, 93 remove aerobus::polynomial < Ring >, 59 aerobus::Quotient < Ring, X >, 65 aerobus::zpz , 88 simplify_t aerobus::polynomial < Ring >, 60
aerobus, 26 aerobus, 28 aerobus, 28 mod_t aerobus::i32, 48 README.md, 93 remove aerobus::polynomial < Ring >, 59 aerobus::Quotient < Ring, X >, 65 aerobus::zpz , 88 simplify_t aerobus::polynomial < Ring >, 59 sin
aerobus, 26 aerobus, 28 aerobus, 28 mod_t      aerobus::i32, 48 README.md, 93      aerobus::i64, 52 remove      aerobus::polynomial < Ring >, 59 aerobus::type_list < Ts >, 70       aerobus::zpz , 88 simplify_t      aerobus::polynomial < Ring >, 59 sin      mul_t      aerobus, 28
aerobus, 26  mod_t  aerobus:i32, 48  aerobus:i64, 52  aerobus::polynomial < Ring >, 59  aerobus::Quotient < Ring, X >, 65  aerobus::zpz , 88  monomial_t  aerobus::polynomial < Ring >, 59  aerobus::polynomial < Ring >, 59  aerobus::polynomial < Ring >, 60  sin  mul_t  aerobus::32, 49  aerobus, 28
aerobus, 26 aerobus, 28 aerobus, 28 mod_t aerobus::i32, 48 README.md, 93 remove aerobus::polynomial < Ring >, 59 aerobus::Quotient < Ring, X >, 65 aerobus::zpz , 88 simplify_t aerobus::polynomial < Ring >, 59 sin aerobus::polynomial < Ring >, 59 sin aerobus::polynomial < Ring >, 59 sin aerobus::i32, 49 aerobus::i64, 52 aerobus, 28 sinh aerobus, 28
aerobus, 26  mod_t  aerobus::i32, 48  aerobus::i64, 52  aerobus::polynomial < Ring >, 59  aerobus::zpz , 88  monomial_t  aerobus::polynomial < Ring >, 59  mul_t  aerobus::i32, 49  aerobus::i32, 49  aerobus::i64, 52  aerobus::polynomial < Ring >, 59  sin  aerobus, 28  sinh  aerobus, 28  SQRT2_fraction
aerobus, 26 aerobus, 28 aerobus, 28 mod_t aerobus::i32, 48 README.md, 93 remove aerobus::polynomial < Ring >, 59 aerobus::Quotient < Ring, X >, 65 aerobus::zpz < $p$ >, 88 simplify_t aerobus::polynomial < Ring >, 59 sin aerobus::polynomial < Ring >, 59 sin aerobus::i32, 49 aerobus::i64, 52 aerobus::polynomial < Ring >, 59 sinh aerobus::i64, 52 aerobus::polynomial < Ring >, 59 sinh aerobus, 28 SQRT2_fraction aerobus::Quotient < Ring, X >, 65 aerobus, 28
aerobus, 26 aerobus, 28 aerobus, 28 mod_t
aerobus, 26 aerobus, 28
aerobus, 26 aerobus, 28 aerobus, 28 mod_t aerobus::i32, 48 aerobus::i64, 52 remove aerobus::Quotient < Ring, $X >$ , 65 aerobus::polynomial < Ring $X >$ , 65 simplify_t aerobus::polynomial < Ring $X >$ , 60 sin aerobus::i32, 49 aerobus::i32, 49 aerobus::i64, 52 aerobus::polynomial < Ring $X >$ , 65 aerobus::Quotient < Ring, $X >$ , 65 aerobus::Quotient < Ring, $X >$ , 65 aerobus::Quotient < Ring, $X >$ , 65 aerobus::polynomial < Ring $X >$ , 65 aerobus::polynomial < Ring, $X >$ , 65 aerobus::po
aerobus, 26 aerobus, 28 $ \begin{array}{ccccccccccccccccccccccccccccccccccc$
aerobus, 26 aerobus, 28 aerobus, 28 mod_t      aerobus::i32, 48 aerobus::i64, 52 remove      aerobus::Quotient < Ring, $X >$ , 65 aerobus::polynomial < Ring >, 59 aerobus::polynomial < Ring >, 59 aerobus::polynomial < Ring >, 60 simplify_t      aerobus::polynomial < Ring >, 59 sin      aerobus::i32, 49 aerobus::i64, 52 aerobus::polynomial < Ring >, 59 aerobus, 28 sinh aerobus::polynomial < Ring >, 59 aerobus, 28 sinh aerobus::Quotient < Ring, $X >$ , 65 aerobus::Quotient < Ring, $X >$ , 65 aerobus::polynomial < Ring >, 59 aerobus, 28 sunh aerobus, 29 aerobus, 29 src/aerobus.h, 93 stirling_signed_t one aerobus, 29
aerobus, 26 aerobus, 28 aerobus, 28 mod_t       aerobus:i32, 48
aerobus, 26  mod_t  aerobus:i32, 48  aerobus:i64, 52  aerobus::polynomial < Ring >, 59  aerobus::quotient < Ring, X >, 65  aerobus::polynomial < Ring >, 59  aerobus::polynomial < Ring >, 59  aerobus::polynomial < Ring >, 59  mul_t  aerobus:i32, 49  aerobus:i64, 52  aerobus::polynomial < Ring >, 59  aerobus:i90;nomial < Ring >, 59  mul_t  aerobus:i04, 52  aerobus:Quotient < Ring >, 59  aerobus:Quotient < Ring, X >, 65  aerobus:Quotient < Ring, X >, 65  aerobus:2pz, 88  mulfractions_t  aerobus, 27  src/aerobus.h, 93  stirling_signed_t  aerobus, 29  stirling_unsigned_t  aerobus, 29  stirling_unsigned_t  aerobus, 29
aerobus, 26 aerobus, 28 aerobus, 28 mod_t aerobus:i32, 48 aerobus:i64, 52 remove aerobus::Quotient Ring, $X >$ , 65 aerobus::polynomial Ring $>$ , 59 aerobus::polynomial Ring $>$ , 59 aerobus::polynomial Ring $>$ , 60 sin aerobus::32, 49 aerobus::polynomial Ring $>$ , 59 aerobus::Quotient Ring, $X >$ , 65 aerobus::Quotient Ring, $X >$ , 65 aerobus::polynomial Ring $>$ , 59 aerobus::polynomial Ring $>$ , 59 aerobus::polynomial Ring $>$ , 59 aerobus::polynomial Ring $>$ , 60 striling_signed_t aerobus, 29 aerobus::32, 49 aerobus::32, 49 aerobus::90 striling_unsigned_t aerobus, 29 aerobus::64, 53 aerobus::polynomial Ring $>$ , 60 strip
aerobus, 26  mod_t  aerobus::i32, 48     aerobus::polynomial
aerobus, 26 aerobus, 28 aerobus, 28 aerobus:i32, 48 aerobus:i64, 52 remove aerobus::polynomial < Ring >, 59 sin aerobus::polynomial < Ring >, 60 sin aerobus::i32, 49 aerobus::polynomial < Ring >, 59 aerobus.:28 SQRT2_fraction aerobus, 28 SQRT3_fraction aerobus, 28 SQRT3_fraction aerobus, 29 striling_signed_t aerobus, 29 striling_unsigned_t aerobus.:polynomial < Ring >, 60 aerobus::polynomial < Ring >, 60 aerobus::Quotient < Ring, X >, 65 aerobus::polynomial < Ring >::val < coeffN >, aerobus::polynomial < Ring >::val < coeffN >; aerobus::po
aerobus, 26 aerobus, 28 aerobus, 28 aerobus:i32, 48 aerobus:i64, 52 remove aerobus::polynomial < Ring >, 59 aerobus::ype_list < Ts >, 70 aerobus::polynomial < Ring >, 59 aerobus::polynomial < Ring >, 59 aerobus::polynomial < Ring >, 59 sim plify_t aerobus::i32, 49 aerobus::i64, 52 aerobus::polynomial < Ring >, 59 aerobus::i04, 52 aerobus::polynomial < Ring >, 59 aerobus::polynomial < Ring >, 65 aerobus::polynomial < Ring >, 65 aerobus::polynomial < Ring >, 60 strip aerobus::polynomial < Ring >, 59 aerobus::polynomial < Ring >, 60 aerobus::polynomial < Ring > :val < coeffN >, aerobus::polynomial < Ring >::val < coeffN >, 78
aerobus, 26  mod_t aerobus::32, 48 aerobus::64, 52 aerobus::polynomial< Ring >, 59 aerobus::polynomial< Ring >, 59 aerobus::polynomial< Ring >, 59 aerobus::polynomial< Ring >, 59 mul_t aerobus::64, 52 aerobus::polynomial< Ring >, 59 mul_t aerobus::i64, 52 aerobus::polynomial< Ring >, 59 aerobus::polynomial< Ring >, 60 aerobus::polynomial< Ring >, 59 aerobus::polynomial< Ring >, 59 aerobus::polynomial< Ring >, 59 aerobus::polynomial< Ring >, 60 aerobus::i64, 53 aerobus::polynomial< Ring >, 60 aerobus::Quotient< Ring, X >, 65 aerobus::Quotient< Ring, X >, 65 aerobus::Quotient< Ring, X >, 65 aerobus::polynomial< Ring >; val< coeffN >, aerobus::polynomial< Ring >::val< coeffN >, aerobus::polynomial< Ring >::val< coeffN >, 78 physicist
aerobus, 26  mod_t aerobus::32, 48 aerobus::64, 52 aerobus::polynomial< Ring >, 59 mul_t aerobus::i32, 49 aerobus::polynomial< Ring >, 59 aerobus::polynomial< Ring >, 60 aerobus::polynomial< Ring >, 59 aerobus::polynomial< Ring >, 60 aerobus::polynomial< Ring >, 60 aerobus::i32, 49 aerobus::i32, 49 aerobus::i64, 53 aerobus::polynomial< Ring >, 60 aerobus::polynomial< Ring >, 60 aerobus::polynomial< Ring >, 60 aerobus::polynomial< Ring >, 80 aerobus::polynomial< Ring >; val< coeffN >, 65 aerobus::polynomial< Ring >:val< coeffN >, 78 physicist aerobus::known_polynomials, 40
aerobus, 26  mod_t     aerobus::i32, 48     aerobus::i64, 52     aerobus::polynomial < Ring >, 59     aerobus::i64, 52     aerobus::i64, 52     aerobus::i64, 52     aerobus::polynomial < Ring >, 59     aerobus::polynomial < Ring >, 65     aerobus::polynomial < Ring >, 65     aerobus::polynomial < Ring >, 60     aerobus::i32, 49     aerobus::i64, 53     aerobus::polynomial < Ring >, 60     aerobus::polynomial < Ring >::val < coeffN > aerobus::polynomial < Ring >::val < coeffN, coeff
aerobus, 26  mod_t aerobus::32, 48 aerobus::64, 52 aerobus::polynomial< Ring >, 59 mul_t aerobus::i32, 49 aerobus::polynomial< Ring >, 59 aerobus::polynomial< Ring >, 60 aerobus::polynomial< Ring >, 59 aerobus::polynomial< Ring >, 60 aerobus::polynomial< Ring >, 60 aerobus::i32, 49 aerobus::i32, 49 aerobus::i64, 53 aerobus::polynomial< Ring >, 60 aerobus::polynomial< Ring >, 60 aerobus::polynomial< Ring >, 60 aerobus::polynomial< Ring >, 80 aerobus::polynomial< Ring >; val< coeffN >, 65 aerobus::polynomial< Ring >:val< coeffN >, 78 physicist aerobus::known_polynomials, 40

```
tail
     aerobus::type_list< Ts >::pop_front, 62
     aerobus::type_list< Ts >::split< index >, 67
tan
     aerobus, 29
tanh
     aerobus, 30
taylor
     aerobus, 30
to_string
    aerobus::i32::val< x >, 74
    aerobus::i64::val< x >, 76
     aerobus::polynomial < Ring >::val < coeffN >, 84
     aerobus::polynomial< Ring >::val< coeffN, coeffs
          >, 79
     aerobus::zpz ::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
```