# 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	22
6.1.2.4 asin	22
6.1.2.5 asinh	22
6.1.2.6 atan	22
6.1.2.7 atanh	23
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	24
6.1.2.12 cosh	24

6.1.2.13 E_fraction	24
6.1.2.14 embed_int_poly_in_fractions_t	24
6.1.2.15 exp	25
6.1.2.16 expm1	25
6.1.2.17 factorial_t	25
6.1.2.18 fpq32	25
6.1.2.19 fpq64	25
6.1.2.20 FractionField	26
6.1.2.21 gcd_t	26
6.1.2.22 geometric_sum	26
6.1.2.23 lnp1	26
6.1.2.24 make_frac_polynomial_t	26
6.1.2.25 make_int_polynomial_t	28
6.1.2.26 make_q32_t	28
6.1.2.27 make_q64_t	28
6.1.2.28 makefraction_t	29
6.1.2.29 mulfractions_t	29
6.1.2.30 pi64	29
6.1.2.31 PI_fraction	29
6.1.2.32 pow_t	29
6.1.2.33 pq64	30
6.1.2.34 q32	30
6.1.2.35 q64	30
6.1.2.36 sin	30
6.1.2.37 sinh	30
6.1.2.38 SQRT2_fraction	31
6.1.2.39 SQRT3_fraction	31
6.1.2.40 stirling_signed_t	31
6.1.2.41 stirling_unsigned_t	31
6.1.2.42 tan	32
6.1.2.43 tanh	32
6.1.2.44 taylor	32
6.1.2.45 vadd_t	32
6.1.2.46 vmul_t	33
6.1.3 Function Documentation	33
6.1.3.1 aligned_malloc()	33
6.1.3.2 field()	33
6.1.4 Variable Documentation	33
6.1.4.1 alternate_v	33
6.1.4.2 bernoulli_v	34
6.1.4.3 combination_v	34
6.1.4.4 factorial_v	34

6.2 aerobus::internal Namespace Reference	35
6.2.1 Detailed Description	38
6.2.2 Typedef Documentation	38
6.2.2.1 make_index_sequence_reverse	38
6.2.2.2 type_at_t	38
6.2.3 Function Documentation	38
6.2.3.1 index_sequence_reverse()	38
6.2.4 Variable Documentation	38
6.2.4.1 is_instantiation_of_v	38
6.3 aerobus::known_polynomials Namespace Reference	39
6.3.1 Detailed Description	39
6.3.2 Typedef Documentation	39
6.3.2.1 bernoulli	39
6.3.2.2 bernstein	40
6.3.2.3 chebyshev_T	40
6.3.2.4 chebyshev_U	40
6.3.2.5 hermite_phys	41
6.3.2.6 hermite_prob	41
6.3.2.7 laguerre	42
6.3.2.8 legendre	42
6.3.3 Enumeration Type Documentation	42
6.3.3.1 hermite_kind	42
7 Concept Documentation	43
7.1 aerobus::IsEuclideanDomain Concept Reference	43
7.1.1 Concept definition	43
7.1.2 Detailed Description	43
7.2 aerobus::IsField Concept Reference	43
7.2.1 Concept definition	43
7.2.2 Detailed Description	44
7.3 aerobus::IsRing Concept Reference	44
7.3.1 Concept definition	44
7.3.2 Detailed Description	44
8 Class Documentation	45
8.1 aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, E > Struct Template Reference	45
8.2 aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable_if_t < (index < 0  index > 0) > > Struct Template Reference	45
8.2.1 Member Typedef Documentation	45
8.2.1.1 type	45
8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> >	_
Struct Template Reference	46
8.3.1 Member Typedef Documentation	46

8.3.1.1 type	46
8.4 aerobus::ContinuedFraction< values > Struct Template Reference	46
8.5 aerobus::ContinuedFraction $<$ a0 $>$ Struct Template Reference	46
8.5.1 Detailed Description	46
8.5.2 Member Typedef Documentation	47
8.5.2.1 type	47
8.5.3 Member Data Documentation	47
8.5.3.1 val	47
8.6 aerobus::ContinuedFraction < a0, rest > Struct Template Reference	47
8.6.1 Detailed Description	47
8.6.2 Member Typedef Documentation	48
8.6.2.1 type	48
8.6.3 Member Data Documentation	48
8.6.3.1 val	48
8.7 aerobus::ConwayPolynomial Struct Reference	48
8.8 aerobus::Embed< Small, Large, E > Struct Template Reference	48
8.9 aerobus::Embed< i32, i64 > Struct Reference	49
8.9.1 Member Typedef Documentation	49
8.9.1.1 type	49
8.10 aerobus::Embed< polynomial< Small >, polynomial< Large > > Struct Template Reference	49
8.10.1 Member Typedef Documentation	49
8.10.1.1 type	49
8.11 aerobus::Embed< q32, q64 > Struct Reference	50
8.11.1 Member Typedef Documentation	50
8.11.1.1 type	50
8.12 aerobus::Embed< Quotient< Ring, X >, Ring > Struct Template Reference	50
8.12.1 Member Typedef Documentation	50
8.12.1.1 type	50
8.13 aerobus::Embed< Ring, FractionField< Ring > > Struct Template Reference	51
8.13.1 Member Typedef Documentation	51
8.13.1.1 type	51
8.14 aerobus::Embed< zpz< x >, i32 > Struct Template Reference	51
8.14.1 Member Typedef Documentation	51
8.14.1.1 type	51
8.15 aerobus::i32 Struct Reference	52
8.15.1 Detailed Description	53
8.15.2 Member Typedef Documentation	53
8.15.2.1 add_t	53
8.15.2.2 div_t	53
8.15.2.3 eq_t	53
8.15.2.4 gcd_t	53
8.15.2.5 gt_t	53

8.15.2.6 inject_constant_t	53
8.15.2.7 inject_ring_t	53
8.15.2.8 inner_type	53
8.15.2.9 lt_t	54
8.15.2.10 mod_t	54
8.15.2.11 mul_t	54
8.15.2.12 one	54
8.15.2.13 pos_t	54
8.15.2.14 sub_t	54
8.15.2.15 zero	54
8.15.3 Member Data Documentation	55
8.15.3.1 eq_v	55
8.15.3.2 is_euclidean_domain	55
8.15.3.3 is_field	55
8.15.3.4 pos_v	55
8.16 aerobus::i64 Struct Reference	55
8.16.1 Detailed Description	56
8.16.2 Member Typedef Documentation	57
8.16.2.1 add_t	57
8.16.2.2 div_t	57
8.16.2.3 eq_t	57
8.16.2.4 gcd_t	57
8.16.2.5 gt_t	57
8.16.2.6 inject_constant_t	57
8.16.2.7 inject_ring_t	57
8.16.2.8 inner_type	58
8.16.2.9 lt_t	58
8.16.2.10 mod_t	58
8.16.2.11 mul_t	58
8.16.2.12 one	58
8.16.2.13 pos_t	58
8.16.2.14 sub_t	58
8.16.2.15 zero	58
8.16.3 Member Data Documentation	59
8.16.3.1 eq_v	59
8.16.3.2 gt_v	59
8.16.3.3 is_euclidean_domain	60
8.16.3.4 is_field	60
8.16.3.5 lt_v	60
8.16.3.6 pos_v	60
8.17 aerobus::is_prime< n > Struct Template Reference	60
8.17.1 Detailed Description	60

8.17.2 Member Data Documentation	61
8.17.2.1 value	61
8.18 aerobus::polynomial < Ring > Struct Template Reference	61
8.18.1 Detailed Description	62
8.18.2 Member Typedef Documentation	63
8.18.2.1 add_t	63
8.18.2.2 derive_t	63
8.18.2.3 div_t	63
8.18.2.4 eq_t	63
8.18.2.5 gcd_t	64
8.18.2.6 gt_t	64
8.18.2.7 inject_constant_t	64
8.18.2.8 inject_ring_t	64
8.18.2.9 lt_t	65
8.18.2.10 mod_t	65
8.18.2.11 monomial_t	65
8.18.2.12 mul_t	65
8.18.2.13 one	66
8.18.2.14 pos_t	66
8.18.2.15 simplify_t	66
8.18.2.16 sub_t	66
8.18.2.17 X	67
8.18.2.18 zero	67
8.18.3 Member Data Documentation	67
8.18.3.1 is_euclidean_domain	67
8.18.3.2 is_field	67
8.18.3.3 pos_v	67
8.19 aerobus::type_list< Ts >::pop_front Struct Reference	67
8.19.1 Detailed Description	68
8.19.2 Member Typedef Documentation	68
8.19.2.1 tail	68
8.19.2.2 type	68
8.20 aerobus::Quotient $<$ Ring, X $>$ Struct Template Reference	68
8.20.1 Detailed Description	69
8.20.2 Member Typedef Documentation	70
8.20.2.1 add_t	70
8.20.2.2 div_t	70
8.20.2.3 eq_t	70
8.20.2.4 inject_constant_t	70
8.20.2.5 inject_ring_t	71
8.20.2.6 mod_t	71
8.20.2.7 mul_t	71

8.20.2.8 one	71
8.20.2.9 pos_t	72
8.20.2.10 zero	72
8.20.3 Member Data Documentation	72
8.20.3.1 eq_v	72
8.20.3.2 is_euclidean_domain	72
8.20.3.3 pos_v	72
8.21 aerobus::type_list< Ts >::split< index > Struct Template Reference	73
8.21.1 Detailed Description	73
8.21.2 Member Typedef Documentation	73
8.21.2.1 head	73
8.21.2.2 tail	73
8.22 aerobus::type_list < Ts > Struct Template Reference	74
8.22.1 Detailed Description	74
8.22.2 Member Typedef Documentation	75
8.22.2.1 at	75
8.22.2.2 concat	75
8.22.2.3 insert	75
8.22.2.4 push_back	75
8.22.2.5 push_front	76
8.22.2.6 remove	76
8.22.3 Member Data Documentation	76
8.22.3.1 length	76
8.23 aerobus::type_list<> Struct Reference	76
8.23.1 Detailed Description	77
8.23.2 Member Typedef Documentation	77
8.23.2.1 concat	77
8.23.2.2 insert	77
8.23.2.3 push_back	77
8.23.2.4 push_front	77
8.23.3 Member Data Documentation	78
8.23.3.1 length	78
8.24 aerobus::i32::val < x > Struct Template Reference	78
8.24.1 Detailed Description	78
8.24.2 Member Typedef Documentation	79
8.24.2.1 enclosing_type	79
8.24.2.2 is_zero_t	79
8.24.3 Member Function Documentation	79
8.24.3.1 eval()	79
8.24.3.2 get()	79
8.24.3.3 to_string()	80
8.24.4 Member Data Documentation	80

8.24.4.1 v	80
8.25 aerobus::i64::val $<$ x $>$ Struct Template Reference	80
8.25.1 Detailed Description	81
8.25.2 Member Typedef Documentation	81
8.25.2.1 enclosing_type	81
8.25.2.2 inner_type	81
8.25.2.3 is_zero_t	81
8.25.3 Member Function Documentation	81
8.25.3.1 eval()	81
8.25.3.2 get()	82
8.25.3.3 to_string()	82
8.25.4 Member Data Documentation	82
8.25.4.1 v	82
8.26 aerobus::polynomial< Ring >::val< coeffN, coeffs > Struct Template Reference	82
8.26.1 Detailed Description	83
8.26.2 Member Typedef Documentation	84
8.26.2.1 aN	84
8.26.2.2 coeff_at_t	84
8.26.2.3 enclosing_type	84
8.26.2.4 is_zero_t	84
8.26.2.5 ring_type	84
8.26.2.6 strip	85
8.26.3 Member Function Documentation	85
8.26.3.1 eval()	85
8.26.3.2 to_string()	85
8.26.4 Member Data Documentation	86
8.26.4.1 degree	86
8.26.4.2 is_zero_v	86
8.27 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference	86
8.27.1 Detailed Description	86
8.27.2 Member Typedef Documentation	87
8.27.2.1 raw_t	87
8.27.2.2 type	87
8.28 aerobus::zpz::val< x > Struct Template Reference	87
8.28.1 Member Typedef Documentation	88
8.28.1.1 enclosing_type	88
8.28.1.2 is_zero_t	88
8.28.2 Member Function Documentation	88
8.28.2.1 eval()	88
8.28.2.2 get()	88
8.28.2.3 to_string()	88
8.28.3 Member Data Documentation	88

99

8.28.3.1 v	. 88
8.29 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference	. 89
8.29.1 Detailed Description	. 89
8.29.2 Member Typedef Documentation	. 90
8.29.2.1 aN	. 90
8.29.2.2 coeff_at_t	. 90
8.29.2.3 enclosing_type	. 90
8.29.2.4 is_zero_t	. 90
8.29.2.5 ring_type	. 90
8.29.2.6 strip	. 90
8.29.3 Member Function Documentation	. 91
8.29.3.1 eval()	. 91
8.29.3.2 to_string()	. 91
8.29.4 Member Data Documentation	. 91
8.29.4.1 degree	. 91
8.29.4.2 is_zero_v	. 91
8.30 aerobus::zpz Struct Template Reference	. 91
8.30.1 Detailed Description	. 93
8.30.2 Member Typedef Documentation	. 93
8.30.2.1 add_t	. 93
8.30.2.2 div_t	. 93
8.30.2.3 eq_t	. 93
8.30.2.4 gcd_t	. 94
8.30.2.5 gt_t	. 94
8.30.2.6 inject_constant_t	. 94
8.30.2.7 inner_type	. 94
8.30.2.8 lt_t	. 94
8.30.2.9 mod_t	. 95
8.30.2.10 mul_t	. 95
8.30.2.11 one	. 95
8.30.2.12 pos_t	. 95
8.30.2.13 sub_t	. 96
8.30.2.14 zero	. 96
8.30.3 Member Data Documentation	. 96
8.30.3.1 eq_v	. 96
8.30.3.2 gt_v	. 96
8.30.3.3 is_euclidean_domain	. 97
8.30.3.4 is_field	. 97
8.30.3.5 lt_v	. 97
8.30.3.6 pos_v	. 97

**9 File Documentation** 

Index

	9.1 README.md File Reference	99
	9.2 src/aerobus.h File Reference	99
	9.3 aerobus.h	99
10	Examples	185
	10.1 QuotientRing	185
	10.2 type_list	185
	10.3 i32::template	185
	10.4 i32::add_t	186
	10.5 i32::sub_t	186
	10.6 i32::mul_t	186
	10.7 i32::div_t	186
	10.8 i32::gt_t	187
	10.9 i32::eq_t	187
	10.10 i32::eq_v	187
	10.11 i32::gcd_t	187
	10.12 i32::pos_t	188
	10.13 i32::pos_v	188
	10.14 i64::template	188
	10.15 i64::add_t	188
	10.16 i64::sub_t	189
	10.17 i64::mul_t	189
	10.18 i64::div_t	189
	10.19 i64::mod_t	189
	10.20 i64::gt_t	190
	10.21 i64::lt_t	190
	10.22 i64::lt_v	190
	10.23 i64::eq_t	190
	10.24 i64::eq_v	191
	10.25 i64::gcd_t	191
	10.26 i64::pos_t	191
	10.27 i64::pos_v	191
	10.28 polynomial	192
	10.29 q32::add_t	192
	10.30 FractionField	192
	10.31 aerobus::ContinuedFraction	192
	10.32 PI_fraction::val	193
	10.33 E_fraction::val	193

195

# 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	35
aerobus::known_polynomials	
Families of well known polynomials such as Hermite or Bernstein	39

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

12 Concept Index

# **Class Index**

# 4.1 Class List

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

$\label{eq:coeffN} $$ \operatorname{aerobus::polynomial} < \operatorname{Ring} > :: \operatorname{val} < \operatorname{coeffN} > :: \operatorname{coeff\_at} < \operatorname{index}, \ E > \dots \dots \dots \dots \\ \operatorname{aerobus::polynomial} < \operatorname{Ring} > :: \operatorname{val} < \operatorname{coeffN} > :: \operatorname{coeff\_at} < \operatorname{index}, \ \operatorname{std::enable\_if\_t} < (\operatorname{index} < 0    \operatorname{index} > 0) > > \dots \\ 45 $$$	45 >
aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable_if_t < (index==0) >>	46
aerobus::ContinuedFraction < values >	46
aerobus::ContinuedFraction< a0 >	40
Specialization for only one coefficient, technically just 'a0'	46
aerobus::ContinuedFraction< a0, rest >	40
	47
Specialization for multiple coefficients (strictly more than one)	
aerobus::ConwayPolynomial	48
aerobus::Embed< Small, Large, E >	48
aerobus::Embed< i32, i64 >	49
aerobus::Embed< polynomial< Small >, polynomial< Large >>	49
aerobus::Embed< q32, q64 >	50
$aerobus:: Embed < Quotient < Ring, X >, Ring > \dots $	50
aerobus::Embed< Ring, FractionField< Ring >>	51
$aerobus::Embed < zpz < x >, i32 > \dots $	51
aerobus::i32	
32 bits signed integers, seen as a algebraic ring with related operations	52
aerobus::i64	
64 bits signed integers, seen as a algebraic ring with related operations	55
aerobus::is_prime< n >	
Checks if n is prime	60
$aerobus::polynomial < Ring > \dots $	61
aerobus::type_list< Ts >::pop_front	
Removes types from head of the list	67
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	68
aerobus::type_list< Ts >::split< index >	
Splits list at index	73
aerobus::type_list< Ts >	
Empty pure template struct to handle type list	74
aerobus::type_list<>	
Specialization for empty type list	76

14 Class Index

aerobus::i32::val< x >	
Values in i32, again represented as types	78
aerobus::i64::val < x >	
Values in i64	80
aerobus::polynomial< Ring >::val< coeffN, coeffs >	
Values (seen as types) in polynomial ring	82
aerobus::Quotient< Ring, X >::val< V >	
Projection values in the quotient ring	86
aerobus::zpz::val< x >	87
aerobus::polynomial< Ring >::val< coeffN >	
Specialization for constants	89
aerobus··zpz< n >	91

# File Index

- 4		 
<b>5</b> 7	FILE	 st
J. I	1 110	 31

Here is a list of all files with brief descriptions:		
src/aerobus.h	99	

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 Embed
- struct Embed< i32, i64 >
- struct Embed< polynomial< Small >, polynomial< Large > >
- struct Embed< q32, q64 >
- struct Embed< Quotient< Ring, X >, Ring >
- struct Embed< Ring, FractionField< Ring > >
- struct Embed< zpz< x >, i32 >
- struct i32

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

struct i64

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

• struct is\_prime

checks if n is prime

- · struct polynomial
- struct Quotient

Quotient ring by the principal ideal generated by 'X' With i32 as Ring and i32::val<2> as X, Quotient is 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.

i64::inject\_constant\_t< q >>>

helper type: make a fraction from numerator and denominator

#### **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<typename v >
  using embed_int_poly_in_fractions_t = typename Embed< polynomial< typename v::ring_type >,
  polynomial < FractionField < typename v::ring_type >>>::template type < v >
     embed a polynomial with integers coefficients into rational coefficients polynomials
• template<int64_t p, int64_t q>
```

using make\_q64\_t = typename q64::template simplify\_t< typename q64::val< i64::inject\_constant\_t< p>,

```
• template<int32_t p, int32_t q>
  using make q32 t = typename q32::template simplify t< typename q32::val< i32::inject constant t< p>,
  i32::inject_constant_t< q >>>
     helper type: make a fraction from numerator and denominator

    template<typename Ring , typename v1 , typename v2 >

  using addfractions t = typename FractionField < Ring >::template add t < v1, v2 >
     helper type : adds two fractions
• template<typename Ring , typename v1 , typename v2 >
  using mulfractions t = typename FractionField < Ring >::template mul t < v1, v2 >
     helper type: multiplies two fractions
• template<typename Ring , auto... xs>
  using make_int_polynomial_t = typename polynomial < Ring >::template val < typename Ring::template
  inject_constant_t< xs >... >
     make a polynomial with coefficients in Ring
• template<typename Ring, auto... xs>
  using make frac polynomial t = typename polynomial < FractionField < Ring > >::template val < typename
  FractionField < Ring >::template inject_constant_t < xs >... >
     make a polynomial with coefficients in FractionField<Ring>
• template<typename T , size_t i>
  using factorial t = typename internal::factorial < T, i >::type
     computes factorial(i), as type
• template<typename T, size_t k, size_t n>
  using combination_t = typename internal::combination< T, k, n >::type
     computes binomial coefficient (k among n) as type
• template<typename T , size t n>
  using bernoulli_t = typename internal::bernoulli < T, n >::type
     nth bernoulli number as type in T
• template<typename T , size_t n>
  using bell_t = typename internal::bell_helper< T, n >::type
     Bell numbers.
• template<typename T , int k>
  using alternate_t = typename internal::alternate < T, k >::type
     (-1)^{\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 >
           \arctan(x)
• template<typename Integers , size_t deg>
    using sin = taylor< Integers, internal::sin_coeff, deg >
           \sin(x)
• template<typename Integers , size_t deg>
    using sinh = taylor < Integers, internal::sh coeff, deg >
• 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,
    1, 2, 1, 2, 1, 2 >
           approximation of
```

#### **Functions**

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

#### **Variables**

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

# 6.1.1 Detailed Description

main namespace for all publicly exposed types or functions

## 6.1.2 Typedef Documentation

#### 6.1.2.1 abs\_t

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

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

#### **Template Parameters**

```
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

	Ring	
	v1	belongs to FractionField <ring></ring>
ĺ	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)
```

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

```
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 embed\_int\_poly\_in\_fractions\_t

embed a polynomial with integers coefficients into rational coefficients polynomials

Lives in polynomial < Fraction Field < Ring >>

Ring	Integers
а	valu in polynomial <ring></ring>

### 6.1.2.15 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

### 6.1.2.16 expm1

```
template<typename Integers , size_t deg> using aerobus::expm1 = typedef typename polynomial<FractionField<Integers> >::template sub_ \leftarrow t< exp<Integers, deg>, typename polynomial<FractionField<Integers> >::one> e^x-1
```

### **Template Parameters**

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

# 6.1.2.17 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	

# 6.1.2.18 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)

### 6.1.2.19 fpq64

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

polynomial with 64 bits rational coefficients

### 6.1.2.20 FractionField

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

# 6.1.2.21 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)
```

### 6.1.2.22 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	taylor approximation degree

# 6.1.2.23 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

### 6.1.2.24 make\_frac\_polynomial\_t

```
template<typename Ring , auto... xs>
using aerobus::make_frac_polynomial_t = typedef typename polynomial<FractionField<Ring> >←
::template val< typename FractionField<Ring>::template inject_constant_t<xs>...>
```

6.1 aerobus Namespace Reference	27
make a polynomial with coefficients in FractionField <ring></ring>	

Ring	integers
xs	values

### 6.1.2.25 make\_int\_polynomial\_t

```
template<typename Ring , auto... xs>
using aerobus::make_int_polynomial_t = typedef typename polynomial<Ring>::template val< typename
Ring::template inject_constant_t<xs>...>
```

make a polynomial with coefficients in Ring

### **Template Parameters**

Ring	integers
xs	coefficients

# 6.1.2.26 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

### 6.1.2.27 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

### 6.1.2.28 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

### 6.1.2.29 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>

### 6.1.2.30 pi64

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

polynomial with 64 bits integers coefficients

### 6.1.2.31 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>
```

### 6.1.2.32 pow\_t

```
template<typename T , typename p , size_t n> using aerobus::pow_t = typedef typename internal::pow<T, p, n>::type p^{\Lambda}n \text{ (as 'val' type in T)}
```

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

### 6.1.2.33 pq64

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

polynomial with 64 bits rationals coefficients

### 6.1.2.34 q32

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

32 bits rationals rationals with 32 bits numerator and denominator

# 6.1.2.35 q64

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

64 bits rationals rationals with 64 bits numerator and denominator

### 6.1.2.36 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

# 6.1.2.37 sinh

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

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

### 6.1.2.38 SQRT2\_fraction

### 6.1.2.39 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> \frac{1}{2}
```

approximation of

### 6.1.2.40 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**

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

# 6.1.2.41 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**

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

### 6.1.2.42 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

### 6.1.2.43 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)
Ī	deg	taylor approximation degree

# 6.1.2.44 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	

# 6.1.2.45 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

# 6.1.2.46 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**



# 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 ( $\operatorname{\textit{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**

```
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)
Т	(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**

T	(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

Т	(aerobus::i64 for example)
i	

# 6.2 aerobus::internal Namespace Reference

internal implementations, subject to breaking changes without notice

### **Classes**

```
    struct _FractionField

    struct FractionField< Ring, std::enable if t< Ring::is euclidean domain > >

    struct is prime

struct _is_prime< 0, i >

    struct _is_prime< 1, i >

• struct _{is}_prime< 2, i >

    struct _is_prime< 3, i >

    struct _is_prime< 5, i >

    struct _is_prime< 7, i >

    struct _is_prime< n, i, std::enable_if_t<(n !=2 &&n !=3 &&n % 2 !=0 &&n % 3==0)>>

    struct _is_prime< n, i, std::enable_if_t<(n !=2 &&n % 2==0)>>

• struct _is_prime< n, i, std::enable_if_t<(n % i==0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&i *i > n)>>
• struct _is_prime< n, i, std::enable_if_t<(n %(i+2) !=0 &&n % i !=0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0
  &&(i *i<=n))> >

    struct _is_prime< n, i, std::enable_if_t<(n %(i+2)==0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&i *i<=n)>

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

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

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

    struct asin coeff

    struct asin_coeff_helper

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

struct asin_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >>
· struct asinh coeff
· struct asinh coeff helper

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

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

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

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

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

    struct bell_helper

    struct bell helper
    T, 0 >
```

struct bell\_helper< T, 1 >

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

    struct bernoulli

    struct bernoulli< T, 0 >

· struct bernoulli coeff

    struct bernoulli helper

    struct bernoulli_helper< T, accum, m, m >

    struct bernstein helper

    struct bernstein_helper< 0, 0, I >

• struct bernstein helper< i, m, l, std::enable if t<(m>0) &&(i>0) &&(i< m)>>

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

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

    struct chebyshev_helper

    struct chebyshev_helper< 1, 0, I >

    struct chebyshev_helper< 1, 1, I >

    struct chebyshev_helper< 2, 0, I >

    struct chebyshev helper< 2, 1, I >

· struct combination

    struct combination helper

    struct combination_helper< T, 0, n >

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

• struct combination helper < T, k, n, std::enable if t<(n >=0 &&k<=(n/2) &&k > 0)> >

    struct cos_coeff

    struct cos_coeff_helper

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

    struct cosh coeff helper

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

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

    struct exp_coeff

· struct factorial

 struct factorial < T, 0 >

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

    struct FractionFieldImpl

    struct FractionFieldImpl< Field, std::enable_if_t< Field::is_field >>

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

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

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

· struct geom_coeff
· struct hermite helper

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

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

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

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

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

    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, l >
 struct laguerre\_helper< 1, l >

```
• 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 pop_front_h

    struct pow

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

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

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

    struct pow scalar

    struct remove h

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

    struct sin_coeff

• struct sin_coeff_helper

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

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

    struct split_h< 0, L1, L2 >

• struct stirling_helper
• struct stirling_helper< T, 0, 0 >
struct stirling_helper< T, 0, n, std::enable_if_t<(n > 0)>>
struct stirling_helper< T, n, 0, std::enable_if_t<(n > 0)> >

    struct stirling_helper< T, n, k, std::enable_if_t<(k > 0) &&(n > 0)> >

· struct tan_coeff
• struct tan coeff helper

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

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

· struct tanh coeff
• struct tanh_coeff_helper

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

    struct tanh coeff helper< T, i, std::enable if t<(i % 2)==0 >>

struct type_at

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

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

### **Typedefs**

```
    template<size_t i, typename... Ts>
        using type_at_t = typename type_at< i, Ts... >::type
    template<std::size_t N>
        using make_index_sequence_reverse = decltype(index_sequence_reverse(std::make_index_sequence< N >{}))
```

### **Functions**

template<std::size\_t... 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
```

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:

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

#include <aerobus.h>

### **Public Types**

using type = aN

# 8.3.1 Member Typedef Documentation

### 8.3.1.1 type

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

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

• src/aerobus.h

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

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

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

• src/aerobus.h

# 8.5 aerobus::ContinuedFraction < a0 > Struct Template Reference

Specialization for only one coefficient, technically just 'a0'.

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

# **Public Types**

using type = typename q64::template inject\_constant\_t< a0 > represented value as aerobus::q64

### **Static Public Attributes**

static constexpr double val = static\_cast<double>(a0)
 represented value as double

# 8.5.1 Detailed Description

```
template<int64_t a0> struct aerobus::ContinuedFraction< a0 >
```

Specialization for only one coefficient, technically just 'a0'.

a0	an integer
	int64_t

# 8.5.2 Member Typedef Documentation

### 8.5.2.1 type

```
template<int64_t a0>
using aerobus::ContinuedFraction< a0 >::type = typename q64::template inject_constant_t<a0>
represented value as aerobus::q64
```

### 8.5.3 Member Data Documentation

### 8.5.3.1 val

```
template<int64_t a0>
constexpr double aerobus::ContinuedFraction< a0 >::val = static_cast<double>(a0) [static],
[constexpr]
```

represented value as double

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

· src/aerobus.h

# 8.6 aerobus::ContinuedFraction< a0, rest... > Struct Template Reference

specialization for multiple coefficients (strictly more than one)

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

# **Public Types**

using type = q64::template add\_t< typename q64::template inject\_constant\_t< a0 >, typename q64
::template div\_t< typename q64::one, typename ContinuedFraction< rest... >::type > >
 represented value as aerobus::q64

### **Static Public Attributes**

static constexpr double val = type::template get<double>()
 represented value as double

# 8.6.1 Detailed Description

```
template<int64_t a0, int64_t... rest> struct aerobus::ContinuedFraction< a0, rest... >
```

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

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

# 8.9 aerobus::Embed < i32, i64 > Struct Reference

#include <aerobus.h>

### **Public Types**

template < typename val >
 using type = i64::val < static\_cast < int64\_t >(val::v)>

# 8.9.1 Member Typedef Documentation

### 8.9.1.1 type

```
template<typename val >
using aerobus::Embed< i32, i64 >::type = i64::val<static_cast<int64_t>(val::v)>
```

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

· src/aerobus.h

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

#include <aerobus.h>

### **Public Types**

template<typename v >
 using type = typename at\_low< v, typename internal::make\_index\_sequence\_reverse< v::degree+1 > >
 ::type

# 8.10.1 Member Typedef Documentation

### 8.10.1.1 type

```
template<typename Small , typename Large >
template<typename v >
using aerobus::Embed< polynomial< Small >, polynomial< Large > >::type = typename at_low<v,
typename internal::make_index_sequence_reverse<v::degree + 1> >::type
```

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

# 8.11 aerobus::Embed < q32, q64 > Struct Reference

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

### **Public Types**

```
    template<typename v >
        using type = make_q64_t< static_cast< int64_t >(v::x::v), static_cast< int64_t >(v::y::v)>
```

# 8.11.1 Member Typedef Documentation

### 8.11.1.1 type

```
template<typename v > using aerobus::Embed< q32, q64 >::type = make_q64_t<static_cast<int64_t>(v::x::v), static_\leftarrow cast<int64_t>(v::y::v)>
```

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

• src/aerobus.h

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

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

# **Public Types**

```
    template<typename val >
        using type = typename val::raw t
```

# 8.12.1 Member Typedef Documentation

### 8.12.1.1 type

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

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

# 8.13 aerobus::Embed< Ring, FractionField< Ring > > Struct Template Reference

#include <aerobus.h>

### **Public Types**

template < typename v >
 using type = typename FractionField < Ring >::template val < v, typename Ring::one >

# 8.13.1 Member Typedef Documentation

### 8.13.1.1 type

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

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

• src/aerobus.h

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

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

### **Public Types**

template<typename val > using type = i32::val< val::v >

### 8.14.1 Member Typedef Documentation

### 8.14.1.1 type

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

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

### 8.15 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

# 8.15.1 Detailed Description

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

# 8.15.2 Member Typedef Documentation

```
8.15.2.1 add_t
template<typename v1 , typename v2 >
using aerobus::i32::add_t = typename add<v1, v2>::type
8.15.2.2 div_t
template<typename v1 , typename v2 >
using aerobus::i32::div_t = typename div<v1, v2>::type
8.15.2.3 eq_t
template<typename v1 , typename v2 >
using aerobus::i32::eq_t = typename eq<v1, v2>::type
8.15.2.4 gcd t
template<typename v1 , typename v2 >
using aerobus::i32::gcd_t = gcd_t<i32, v1, v2>
8.15.2.5 gt_t
template<typename v1 , typename v2 >
using aerobus::i32::gt_t = typename gt<v1, v2>::type
8.15.2.6 inject_constant_t
template<auto x>
using aerobus::i32::inject_constant_t = val<static_cast<int32_t>(x)>
8.15.2.7 inject_ring_t
{\tt template}{<}{\tt typename}\ {\tt v}\ >
using aerobus::i32::inject_ring_t = v
8.15.2.8 inner_type
```

using aerobus::i32::inner\_type = int32\_t

### 8.15.2.9 lt\_t

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

# 8.15.2.10 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 i32
v2	a value in i32

### 8.15.2.11 mul\_t

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

### 8.15.2.12 one

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

constant one

# 8.15.2.13 pos\_t

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

### 8.15.2.14 sub\_t

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

### 8.15.2.15 zero

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

constant zero

# 8.15.3 Member Data Documentation

### 8.15.3.1 eq v

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

### 8.15.3.2 is\_euclidean\_domain

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

integers are an euclidean domain

# 8.15.3.3 is\_field

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

integers are not a field

### 8.15.3.4 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

# 8.16 aerobus::i64 Struct Reference

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

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

### Classes

struct val

values in i64

### **Public Types**

```
• using inner_type = int64_t
     type of represented values

    template<auto x>

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

    template<typename v >

  using inject ring t = v
     injects a value used for internal consistency and quotient rings implementations for example i64::inject_ring_t<i64::val<1>>
     -> i64::val<1>
using zero = val< 0 >
     constant zero

    using one = val< 1 >

     constant one
• template<typename v1 , typename v2 >
 using add_t = typename add< v1, v2 >::type
• template<typename v1 , typename v2 >
  using sub t = typename sub < v1, v2 >::type
• template<typename v1 , typename v2 >
  using mul_t = typename mul < v1, v2 >::type
• template<typename v1 , typename v2 >
  using div_t = typename div < v1, v2 >::type
• template<typename v1 , typename v2 >
  using mod_t = typename remainder < v1, v2 >::type
• template<typename v1 , typename v2 >
 using gt t = typename gt < v1, v2 >::type
• template<typename v1 , typename v2 >
  using lt_t = typename lt< v1, v2 >::type
• template<typename v1 , typename v2 >
 using eq_t = typename eq< v1, v2 >::type

    template<typename v1 , typename v2 >

  using gcd_t = gcd_t < i64, v1, v2 >

    template<typename v >

  using pos t = typename pos< v >::type
```

### Static Public Attributes

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

# 8.16.1 Detailed Description

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

# 8.16.2 Member Typedef Documentation

# 8.16.2.1 add t template<typename v1 , typename v2 >using aerobus::i64::add\_t = typename add<v1, v2>::type 8.16.2.2 div t template<typename v1 , typename v2 >using aerobus::i64::div\_t = typename div<v1, v2>::type 8.16.2.3 eq\_t template<typename v1 , typename v2 >using aerobus::i64::eq\_t = typename eq<v1, v2>::type 8.16.2.4 gcd\_t template<typename v1 , typename v2 >using aerobus:: $i64::gcd_t = gcd_t < i64$ , v1, v2> 8.16.2.5 gt\_t template<typename v1 , typename v2 >using aerobus::i64::gt\_t = typename gt<v1, v2>::type 8.16.2.6 inject\_constant\_t template<auto x> using aerobus::i64::inject\_constant\_t = val<static\_cast<int64\_t>(x)> 8.16.2.7 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** 

a value in i64

```
8.16.2.8 inner_type
```

```
using aerobus::i64::inner_type = int64_t
type of represented values
8.16.2.9 lt_t
template<typename v1 , typename v2 >
using aerobus::i64::lt_t = typename lt<v1, v2>::type
8.16.2.10 mod t
template<typename v1 , typename v2 >
using aerobus::i64::mod_t = typename remainder<v1, v2>::type
8.16.2.11 mul_t
template<typename v1 , typename v2 >
using aerobus::i64::mul_t = typename mul<v1, v2>::type
8.16.2.12 one
using aerobus::i64::one = val<1>
constant one
8.16.2.13 pos_t
template<typename v >
using aerobus::i64::pos_t = typename pos<v>::type
8.16.2.14 sub_t
template<typename v1 , typename v2 >
using aerobus::i64::sub_t = typename sub<v1, v2>::type
8.16.2.15 zero
using aerobus::i64::zero = val<0>
constant zero
```

# 8.16.3 Member Data Documentation

# 8.16.3.1 eq\_v

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

8.16.3.2 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

### 8.16.3.3 is\_euclidean\_domain

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

integers are an euclidean domain

### 8.16.3.4 is field

integers are not a field

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

# 8.16.3.5 lt\_v

### 8.16.3.6 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

# 8.17 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

# 8.17.1 Detailed Description

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

checks if n is prime

**Template Parameters** 

```
n
```

#### 8.17.2 Member Data Documentation

#### 8.17.2.1 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

## 8.18 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

## 8.18.1 Detailed Description

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

polynomial with coefficients in Ring Ring must be an integral domain

## 8.18.2 Member Typedef Documentation

#### 8.18.2.1 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	

## 8.18.2.2 derive\_t

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

## derivation operator

## **Template Parameters**



## 8.18.2.3 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	

## 8.18.2.4 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	

#### 8.18.2.5 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	

## 8.18.2.6 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	

## 8.18.2.7 inject\_constant\_t

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

## 8.18.2.8 inject\_ring\_t

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

#### 8.18.2.9 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	

## 8.18.2.10 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	

#### 8.18.2.11 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	

## 8.18.2.12 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	

#### 8.18.2.13 one

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

constant one

#### 8.18.2.14 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**



## 8.18.2.15 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** 



#### 8.18.2.16 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	

#### 8.18.2.17 X

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

#### 8.18.2.18 zero

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

constant zero

## 8.18.3 Member Data Documentation

#### 8.18.3.1 is\_euclidean\_domain

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

## 8.18.3.2 is\_field

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

## 8.18.3.3 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

## 8.19 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
```

## 8.19.1 Detailed Description

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

removes types from head of the list

## 8.19.2 Member Typedef Documentation

#### 8.19.2.1 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

## 8.19.2.2 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

## 8.20 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
```

## 8.20.1 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>

## 8.20.2 Member Typedef Documentation

#### 8.20.2.1 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**

ν	/1	a value in quotient ring
V	/2	a value in quotient ring

## 8.20.2.2 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

## 8.20.2.3 eq\_t

```
template<typename Ring , typename X >
template<typename v1 , typename v2 >
using aerobus::Quotient< Ring, X >::eq_t = typename Ring::template eq_t<typename v1::type,
typename v2::type>
```

## equality operator (as type)

## **Template Parameters**

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

#### 8.20.2.4 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>
>
```

## 8.20.2.5 inject\_ring\_t

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

## 8.20.2.6 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 qu	a value in quotient ring
v2	a value in quotient ring

## 8.20.2.7 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		a value in quotient ring
	v2	a value in quotient ring

## 8.20.2.8 one

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

one

#### 8.20.2.9 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
```

#### 8.20.2.10 zero

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

zero value

#### 8.20.3 Member Data Documentation

#### 8.20.3.1 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

## $8.20.3.2 \quad is\_euclidean\_domain$

```
template<typename Ring , typename X >
constexpr bool aerobus::Quotient< Ring, X >::is_euclidean_domain = true [static], [constexpr]
quotien rings are euclidean domain
```

## 8.20.3.3 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

# 8.21 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

## 8.21.1 Detailed Description

```
template<typename... Ts>
template<size_t index>
struct aerobus::type_list< Ts >::split< index >
splits list at index
Template Parameters
```

## 8.21.2 Member Typedef Documentation

## 8.21.2.1 head

index

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

#### 8.21.2.2 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

## 8.22 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
```

## 8.22.1 Detailed Description

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

Empty pure template struct to handle type list.

## 8.22.2 Member Typedef Documentation

#### 8.22.2.1 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** 

#### 8.22.2.2 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** 



## 8.22.2.3 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	

## 8.22.2.4 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
---

## 8.22.2.5 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** 



#### 8.22.2.6 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
```

## 8.22.3 Member Data Documentation

## 8.22.3.1 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

## 8.23 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

## 8.23.1 Detailed Description

specialization for empty type list

## 8.23.2 Member Typedef Documentation

#### 8.23.2.1 concat

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

## 8.23.2.2 insert

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

#### 8.23.2.3 push\_back

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

## 8.23.2.4 push\_front

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

## 8.23.3 Member Data Documentation

#### 8.23.3.1 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

## 8.24 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

## 8.24.1 Detailed Description

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

values in i32, again represented as types

**Template Parameters** 

```
x an actual integer
```

## 8.24.2 Member Typedef Documentation

## 8.24.2.1 enclosing\_type

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

Enclosing ring type.

## 8.24.2.2 is\_zero\_t

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

is value zero

## 8.24.3 Member Function Documentation

## 8.24.3.1 eval()

cast x into valueRing

**Template Parameters** 

```
valueRing double for example
```

#### 8.24.3.2 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
```

#### 8.24.3.3 to\_string()

string representation of value

#### 8.24.4 Member Data Documentation

## 8.24.4.1 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

## 8.25 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

## 8.25.1 Detailed Description

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

Template Parameters

x an actual integer
```

## 8.25.2 Member Typedef Documentation

## 8.25.2.1 enclosing\_type

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

## 8.25.2.2 inner\_type

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

type of represented values

#### 8.25.2.3 is\_zero\_t

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

## 8.25.3 Member Function Documentation

## 8.25.3.1 eval()

cast value in valueRing

## **Template Parameters**

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

#### 8.25.3.2 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)
```

## 8.25.3.3 to\_string()

string representation

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

## 8.25.4 Member Data Documentation

## 8.25.4.1 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

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

values (seen as types) in polynomial ring

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

## **Public Types**

```
    using ring_type = Ring
        ring coefficients live in
    using enclosing_type = polynomial < Ring >
        enclosing ring type
    using aN = coeffN
        heavy weight coefficient (non zero)
    using strip = val < coeffs... >
        remove largest coefficient
    using is_zero_t = std::bool_constant < (degree==0) &&(aN::is_zero_t::value) >
        true_type if polynomial is constant zero
    template < size_t index >
        using coeff_at_t = typename coeff_at < index >::type
```

#### **Static Public Member Functions**

type of coefficient at index

- 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

## 8.26.1 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

## 8.26.2 Member Typedef Documentation

#### 8.26.2.1 aN

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

heavy weight coefficient (non zero)

#### 8.26.2.2 coeff\_at\_t

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
template<size_t index>
using aerobus::polynomial< Ring >::val< coeffN, coeffs >::coeff_at_t = typename coeff_\to at<index>::type
```

type of coefficient at index

#### **Template Parameters**

```
index
```

## 8.26.2.3 enclosing\_type

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

enclosing ring type

## 8.26.2.4 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

## 8.26.2.5 ring\_type

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

ring coefficients live in

## 8.26.2.6 strip

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

remove largest coefficient

#### 8.26.3 Member Function Documentation

#### 8.26.3.1 eval()

evaluates polynomial seen as a function operating on ValueRing

## **Template Parameters**

valueRing	usually float or double
-----------	-------------------------

#### **Parameters**

```
x value
```

## Returns

P(x)

## 8.26.3.2 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
```

## 8.26.4 Member Data Documentation

#### 8.26.4.1 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

## 8.26.4.2 is\_zero\_v

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
constexpr bool aerobus::polynomial< Ring >::val< coeffN, coeffs >::is_zero_v = is_zero_t ↔
::value [static], [constexpr]
```

true if polynomial is constant zero

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

• src/aerobus.h

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

projection values in the quotient ring

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

## **Public Types**

- using raw t = V
- using type = abs\_t< typename Ring::template mod\_t< V, X >>

## 8.27.1 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'
```

## 8.27.2 Member Typedef Documentation

#### 8.27.2.1 raw t

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

#### 8.27.2.2 type

```
template<typename Ring , typename X >
template<typename V >
using aerobus::Quotient< Ring, X >::val< V >::type = abs_t<typename Ring::template mod_t<V,
X> >
```

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

• src/aerobus.h

## 8.28 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

## 8.28.1 Member Typedef Documentation

## 8.28.1.1 enclosing\_type

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

8.28.1.2 is_zero_t
```

## 8.28.2 Member Function Documentation

## 8.28.2.1 eval()

template<int32\_t p>
template<int32\_t x>

## 8.28.2.2 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]
```

## 8.28.2.3 to\_string()

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

#### 8.28.3 Member Data Documentation

## 8.28.3.1 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

## 8.29 aerobus::polynomial< Ring >::val< coeffN > Struct Template Reference

specialization for constants

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

#### Classes

- · struct coeff at
- struct coeff\_at< index, std::enable\_if\_t<(index<0||index > 0)>>
- struct coeff at< index, std::enable if t<(index==0)>>

## **Public Types**

```
• using ring_type = Ring
```

ring coefficients live in

using enclosing\_type = polynomial< Ring >

enclosing ring type

- using aN = coeffN
- using strip = val< coeffN >
- using is\_zero\_t = std::bool\_constant< aN::is\_zero\_t::value >
- template<size\_t index>

```
using coeff_at_t = typename coeff_at< index >::type
```

#### **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

## 8.29.1 Detailed Description

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

**Template Parameters** 

```
coeffN
```

## 8.29.2 Member Typedef Documentation

#### 8.29.2.1 aN

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

## 8.29.2.2 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
```

#### 8.29.2.3 enclosing\_type

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

## 8.29.2.4 is\_zero\_t

enclosing ring type

```
template<typename Ring >
template<typename coeffN >
using aerobus::polynomial< Ring >::val< coeffN >::is_zero_t = std::bool_constant<aN::is_\top zero_t::value>
```

## 8.29.2.5 ring\_type

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

ring coefficients live in

#### 8.29.2.6 strip

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

#### 8.29.3 Member Function Documentation

#### 8.29.3.1 eval()

#### 8.29.3.2 to\_string()

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

#### 8.29.4 Member Data Documentation

#### 8.29.4.1 degree

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

## 8.29.4.2 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

## 8.30 aerobus::zpz Struct Template Reference

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

#### **Classes**

struct val

## **Public Types**

```
• using inner_type = int32_t

    template<auto x>

  using inject_constant_t = val< static_cast< int32_t >(x)>
using zero = val< 0 >
using one = val< 1 >
• template<typename v1 , typename v2 >
  using add_t = typename add< v1, v2 >::type
     addition operator
• template<typename v1 , typename v2 >
  using sub_t = typename sub< v1, v2 >::type
     substraction operator

    template<typename v1 , typename v2 >

  using mul t = typename mul < v1, v2 >::type
     multiplication operator
• template<typename v1 , typename v2 >
  using div t = typename div < v1, v2 >::type
     division operator
• template<typename v1 , typename v2 >
  using mod_t = typename remainder < v1, v2 >::type
     modulo operator
• template<typename v1 , typename v2 >
  using gt_t = typename gt < v1, v2 >::type
     strictly greater operator (type)
• template<typename v1 , typename v2 >
  using It_t = typename It< v1, v2 >::type
     strictly smaller operator (type)
• template<typename v1 , typename v2 >
  using eq_t = typename eq< v1, v2 >::type
     equality operator (type)
• template<typename v1 , typename v2 >
  using gcd_t = gcd_t < i32, v1, v2 >
     greatest common divisor
template<typename v1 >
  using pos_t = typename pos< v1 >::type
     positivity operator (type)
```

#### Static Public Attributes

```
    static constexpr bool is_field = is_prime::value
    static constexpr bool is_euclidean_domain = true
    template<typename v1 , typename v2 >
        static constexpr bool gt_v = gt_t<v1, v2>::value
            strictly greater operator (booleanvalue)
    template<typename v1 , typename v2 >
        static constexpr bool lt_v = lt_t<v1, v2>::value
            strictly smaller operator (booleanvalue)
    template<typename v1 , typename v2 >
        static constexpr bool eq_v = eq_t<v1, v2>::value
            equality operator (booleanvalue)
    template<typename v >
        static constexpr bool pos_v = pos_t<v>::value
            positivity operator (boolean value)
```

## 8.30.1 Detailed Description

```
\label{eq:continuous_to_p} \begin{split} \text{template} &< \text{int32\_t p} > \\ \text{struct aerobus::zpz} &< \text{p} > \end{split}
```

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

## 8.30.2 Member Typedef Documentation

## 8.30.2.1 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

## 8.30.2.2 div\_t

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

## division operator

## **Template Parameters**

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

## 8.30.2.3 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

## 8.30.2.4 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

## 8.30.2.5 gt\_t

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

## strictly greater operator (type)

#### **Template Parameters**

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

## 8.30.2.6 inject\_constant\_t

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

## 8.30.2.7 inner\_type

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

## 8.30.2.8 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

## 8.30.2.9 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

## 8.30.2.10 mul\_t

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

## multiplication operator

#### **Template Parameters**

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

## 8.30.2.11 one

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

## 8.30.2.12 pos\_t

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

## positivity operator (type)

## **Template Parameters**

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

#### 8.30.2.13 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

#### 8.30.2.14 zero

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

## 8.30.3 Member Data Documentation

## 8.30.3.1 eq\_v

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

## equality operator (booleanvalue)

## **Template Parameters**

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

## 8.30.3.2 gt\_v

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

strictly greater operator (booleanvalue)

## **Template Parameters**

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

## 8.30.3.3 is\_euclidean\_domain

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

## 8.30.3.4 is\_field

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

## 8.30.3.5 lt\_v

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

strictly smaller operator (booleanvalue)

## **Template Parameters**

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

## 8.30.3.6 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

98 Class Documentation

# **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__
```

```
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>;
08000
               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
               };
```

```
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} >
                             struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)» {
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
```

```
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 // embedding
00327 namespace aerobus {
00328
         template<typename Small, typename Large, typename E = void>
00329
          struct Embed;
00330 }
        // namespace aerobus
00331
00332 namespace aerobus {
00337
         template<typename Ring, typename X>
00338
          requires IsRing<Ring>
00339
          struct Quotient {
00342
              template <typename V>
00343
              struct val {
00344
              public:
00345
                 using raw_t = V;
00346
                  using type = abs_t<typename Ring::template mod_t<V, X>>;
00347
00348
00350
              using zero = val<typename Ring::zero>;
00351
00353
              using one = val<typename Ring::one>;
00354
00358
              template<typename v1, typename v2>
00359
              using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00360
00364
              template<typename v1, typename v2>
00365
              using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00366
00370
              template<typename v1, typename v2>
00371
              using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00372
00376
              template<typename v1, typename v2>
00377
              using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00378
00382
              template<typename v1, typename v2>
00383
              using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00384
00388
              template<typename v1, typename v2> \,
00389
              static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00390
00394
              template<typename v1>
00395
              using pos_t = std::true_type;
00396
00400
              template<typename v>
              static constexpr bool pos_v = pos_t<v>::value;
00401
00402
00404
              static constexpr bool is_euclidean_domain = true;
00405
00411
              template<auto x>
00412
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00413
00419
              template<typename v>
00420
              using inject_ring_t = val<v>;
00421
00422
00423
          template<typename Ring, typename X>
          struct Embed<Quotient<Ring, X>, Ring> {
00424
              template<typename val>
00425
00426
              using type = typename val::raw_t;
00427
00428 }
        // namespace aerobus
00429
00430 // type list
00431 namespace aerobus {
00433
         template <typename... Ts>
00434
         struct type_list;
00435
00436
          namespace internal {
00437
              template <typename T, typename... Us>
00438
              struct pop_front_h {
00439
                  using tail = type_list<Us...>;
00440
                  using head = T;
00441
00442
00443
              template <size_t index, typename L1, typename L2>
              struct split_h {
00444
00445
              private:
00446
                  static_assert(index <= L2::length, "index ouf of bounds");</pre>
00447
                  using a = typename L2::pop_front::type;
                  using b = typename L2::pop_front::tail;
00448
00449
                  using c = typename L1::template push_back<a>;
00450
00451
               public:
```

```
using head = typename split_h<index - 1, c, b>::head;
using tail = typename split_h<index - 1, c, b>::tail;
00453
00454
                };
00455
00456
                template <typename L1, typename L2>
struct split_h<0, L1, L2> {
00457
                    using head = L1;
00458
00459
                     using tail = L2;
00460
                } ;
00461
00462
                template <size_t index, typename L, typename T>
00463
                struct insert h {
00464
                    static_assert(index <= L::length, "index ouf of bounds");</pre>
00465
                    using s = typename L::template split<index>;
00466
                    using left = typename s::head;
                    using right = typename s::tail;
00467
                    using 11 = typename left::template push_back<T>;
using type = typename l1::template concat<right>;
00468
00469
00471
00472
                template <size_t index, typename L>
00473
                struct remove_h {
00474
                    using s = typename L::template split<index>;
using left = typename s::head;
using right = typename s::tail;
00475
00476
00477
                    using rr = typename right::pop_front::tail;
00478
                     using type = typename left::template concat<rr>;
00479
           } // namespace internal
00480
00481
00485
           template <typename... Ts>
00486
           struct type_list {
00487
00488
                template <typename T>
00489
                struct concat_h;
00490
00491
                template <typename... Us>
00492
                struct concat_h<type_list<Us...» {
00493
                    using type = type_list<Ts..., Us...>;
00494
00495
            public:
00496
00498
               static constexpr size_t length = sizeof...(Ts);
00499
00502
                template <typename T>
00503
                using push_front = type_list<T, Ts...>;
00504
00507
                template <size_t index>
00508
                using at = internal::type_at_t<index, Ts...>;
00509
00511
                struct pop_front {
                    using type = typename internal::pop_front_h<Ts...>::head;
using tail = typename internal::pop_front_h<Ts...>::tail;
00513
00515
00516
                };
00517
00520
                template <typename T>
00521
                using push_back = type_list<Ts..., T>;
00522
00525
                template <typename U>
00526
                using concat = typename concat_h<U>::type;
00527
00530
                template <size_t index>
00531
                struct split {
00532
00533
                    using inner = internal::split_h<index, type_list<>, type_list<Ts...»;</pre>
00534
00535
                    using head = typename inner::head;
00536
                    using tail = typename inner::tail;
00537
00538
00539
00543
                template <typename T, size_t index>
00544
                using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00545
                template <size_t index>
using remove = typename internal::remove_h<index, type_list<Ts...»::type;</pre>
00548
00549
00550
00551
00553
           template <>
00554
           struct type_list<> {
00555
                static constexpr size t length = 0;
00557
                template <typename T>
00558
                using push_front = type_list<T>;
00559
                template <typename T>
00560
                using push_back = type_list<T>;
00561
```

```
00563
              template <typename U>
00564
              using concat = U;
00565
00566
              // TODO(jewave): assert index == 0
              template <typename T, size_t index>
using insert = type_list<T>;
00567
00569
00570 } // namespace aerobus
00571
00572 // i32
00573 namespace aerobus {
00575
          struct i32 {
00576
              using inner_type = int32_t;
00579
              template<int32_t x>
00580
              struct val {
                  using enclosing_type = i32;
00582
                  static constexpr int32_t v = x;
00584
00585
00588
                  template<typename valueType>
00589
                  static constexpr INLINED DEVICE valueType get() { return static_cast<valueType>(x); }
00590
00592
                  using is_zero_t = std::bool_constant<x == 0>;
00593
00595
                  static std::string to_string() {
00596
                     return std::to_string(x);
00597
00598
00601
                  template<typename valueRing>
                  static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
00602
00603
                      return static cast<valueRing>(x);
00604
00605
00606
              using zero = val<0>;
using one = val<1>;
00608
00610
              static constexpr bool is_field = false;
00612
00614
              static constexpr bool is_euclidean_domain = true;
00618
              template<auto x>
00619
              using inject_constant_t = val<static_cast<int32_t>(x)>;
00620
00621
              template<typename v>
00622
              using inject_ring_t = v;
00623
           private:
00624
00625
              template<typename v1, typename v2>
00626
              struct add {
                  using type = val<v1::v + v2::v>;
00627
00628
00629
00630
              template<typename v1, typename v2>
00631
00632
                  using type = val<v1::v - v2::v>;
00633
00634
              template<typename v1, typename v2>
00635
              struct mul {
00637
                  using type = val<v1::v* v2::v>;
00638
00639
00640
              template<typename v1, typename v2>
00641
              struct div {
00642
                  using type = val<v1::v / v2::v>;
00643
00644
00645
              template<typename v1, typename v2>
00646
              struct remainder {
                  using type = val<v1::v % v2::v>;
00647
00648
00650
              template<typename v1, typename v2>
00651
00652
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00653
00654
00655
              template<typename v1, typename v2>
00656
00657
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00658
              };
00659
00660
              template<typename v1, typename v2>
00661
              struct eq {
00662
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00663
00664
00665
              template<typename v1>
00666
              struct pos {
```

```
00667
                  using type = std::bool_constant<(v1::v > 0)>;
00668
              };
00669
00670
           public:
00676
              template<typename v1, typename v2>
00677
              using add_t = typename add<v1, v2>::type;
00678
00684
              template<typename v1, typename v2>
00685
              using sub_t = typename sub<v1, v2>::type;
00686
00692
              template<typename v1, typename v2>
00693
              using mul_t = typename mul<v1, v2>::type;
00694
00700
              template<typename v1, typename v2>
00701
              using div_t = typename div<v1, v2>::type;
00702
00708
              template<typename v1, typename v2>
00709
              using mod_t = typename remainder<v1, v2>::type;
00710
00716
              template<typename v1, typename v2>
00717
              using gt_t = typename gt<v1, v2>::type;
00718
00724
              template<typename v1, typename v2>
00725
              using lt_t = typename lt<v1, v2>::type;
00726
00732
              template<typename v1, typename v2>
00733
              using eq_t = typename eq<v1, v2>::type;
00734
00739
              template<typename v1, typename v2>
              static constexpr bool eq_v = eq_t<v1, v2>::value;
00740
00741
              template<typename v1, typename v2>
using gcd_t = gcd_t<i32, v1, v2>;
00747
00748
00749
00754
              {\tt template}{<}{\tt typename}\ {\tt v}{>}
00755
              using pos_t = typename pos<v>::type;
00756
00761
              template<typename v>
00762
              static constexpr bool pos_v = pos_t<v>::value;
00763
00764 } // namespace aerobus
00765
00766 // i64
00767 namespace aerobus {
00769
         struct i64 {
00771
             using inner_type = int64_t;
00774
              template<int64_t x>
00775
              struct val {
00777
                  using inner_type = int32_t;
                  using enclosing_type = i64;
00779
00781
                  static constexpr int64_t v = x;
00782
00785
                  template<typename valueType>
                  static constexpr DEVICE INLINED valueType get() {
00786
00787
                       return static_cast<valueType>(x);
00788
                  }
00789
00791
                  using is_zero_t = std::bool_constant<x == 0>;
00792
00794
                  static std::string to_string() {
00795
                      return std::to_string(x);
00796
                  }
00797
00800
                  template<typename valueRing>
00801
                  static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
00802
                       return static_cast<valueRing>(x);
00803
00804
              };
00805
00809
              template<auto x>
00810
              using inject_constant_t = val<static_cast<int64_t>(x)>;
00811
00816
              template<typename v>
00817
              using inject_ring_t = v;
00818
00820
              using zero = val<0>;
00822
              using one = val<1>;
00824
              static constexpr bool is_field = false;
00826
              static constexpr bool is_euclidean_domain = true;
00827
00828
           private:
00829
              template<typename v1, typename v2>
00830
              struct add {
00831
                  using type = val<v1::v + v2::v>;
00832
00833
00834
              template<tvpename v1, tvpename v2>
```

```
00835
              struct sub {
                  using type = val<v1::v - v2::v>;
00836
00837
              };
00838
              template<typename v1, typename v2> ^{\circ}
00839
00840
              struct mul {
00841
                  using type = val<v1::v* v2::v>;
00842
00843
00844
              template<typename v1, typename v2>
00845
              struct div {
                 using type = val<v1::v / v2::v>;
00846
00847
00848
00849
              template<typename v1, typename v2>
00850
              struct remainder {
                  using type = val<v1::v% v2::v>;
00851
00852
00853
00854
              template<typename v1, typename v2>
00855
00856
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00857
00858
00859
              template<typename v1, typename v2>
00860
              struct lt {
00861
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00862
00863
00864
              template<typename v1, typename v2>
00865
              struct eq {
00866
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00867
00868
00869
              template<typename v>
00870
              struct pos {
00871
                  using type = std::bool_constant<(v::v > 0)>;
00873
00874
           public:
00879
              template<typename v1, typename v2> \,
00880
              using add_t = typename add<v1, v2>::type;
00881
00886
              template<typename v1, typename v2>
00887
              using sub_t = typename sub<v1, v2>::type;
00888
00893
              template<typename v1, typename v2>
00894
              using mul_t = typename mul<v1, v2>::type;
00895
00901
              template<typename v1, typename v2>
00902
              using div_t = typename div<v1, v2>::type;
00903
00908
              template<typename v1, typename v2>
00909
              using mod_t = typename remainder<v1, v2>::type;
00910
00916
              template<typename v1, typename v2>
00917
              using gt_t = typename gt<v1, v2>::type;
00918
00923
              template<typename v1, typename v2>
00924
              static constexpr bool gt_v = gt_t<v1, v2>::value;
00925
00931
              template<typename v1, typename v2>
00932
              using lt_t = typename lt<v1, v2>::type;
00933
00939
              template<typename v1, typename v2>
00940
              static constexpr bool lt_v = lt_t<v1, v2>::value;
00941
00947
              template<typename v1, typename v2>
00948
              using eq_t = typename eq<v1, v2>::type;
00949
00955
              template<typename v1, typename v2>
00956
              static constexpr bool eq_v = eq_t<v1, v2>::value;
00957
00963
              template<typename v1, typename v2> ^{\circ}
00964
              using gcd_t = gcd_t < i64, v1, v2>;
00965
00970
              template<typename v>
00971
              using pos_t = typename pos<v>::type;
00972
00977
              template<tvpename v>
00978
              static constexpr bool pos_v = pos_t<v>::value;
00979
          };
00980
00981
          template<>
00982
          struct Embed<i32, i64> {
00983
              template<typename val>
00984
              using type = i64::val<static_cast<int64_t>(val::v)>;
```

```
00986 } // namespace aerobus
00987
00988 // z/pz
00989 namespace aerobus {
00994
         template<int32_t p>
          struct zpz {
00996
             using inner_type = int32_t;
00997
              template<int32_t x>
00998
              struct val {
                 using enclosing_type = zpz;
01000
                  static constexpr int32_t v = x % p;
01002
01003
01004
                 template<typename valueType>
01005
                  static constexpr DEVICE INLINED valueType get() { return static_cast<valueType>(x % p); }
01006
                  using is_zero_t = std::bool_constant<x% p == 0>;
01007
                  static std::string to_string() {
01008
                     return std::to_string(x % p);
01009
01010
                  }
01011
01012
                  template<typename valueRing>
                  static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
01013
                      return static_cast<valueRing>(x % p);
01014
01015
01016
              };
01017
01018
              template<auto x>
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01019
01020
01021
              using zero = val<0>:
01022
              using one = val<1>;
01023
              static constexpr bool is_field = is_prime::value;
01024
              static constexpr bool is_euclidean_domain = true;
01025
01026
           private:
              template<typename v1, typename v2>
01027
              struct add {
01029
                 using type = val<(v1::v + v2::v) % p>;
01030
01031
              template<typename v1, typename v2>
01032
01033
              struct sub {
01034
                 using type = val<(v1::v - v2::v) % p>;
01035
01036
01037
              template<typename v1, typename v2>
01038
              struct mul {
                 using type = val<(v1::v* v2::v) % p>;
01039
01040
01041
01042
              template<typename v1, typename v2>
01043
              struct div {
01044
                using type = val<(v1::v% p) / (v2::v % p)>;
01045
01046
01047
              template<typename v1, typename v2>
01048
              struct remainder {
01049
                 using type = val<(v1::v% v2::v) % p>;
01050
01051
              template<typename v1, typename v2>
01052
01053
              struct qt {
01054
                 using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01055
01056
01057
              template<typename v1, typename v2>
01058
              struct lt {
                 using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01059
01060
01061
01062
              template<typename v1, typename v2>
01063
              struct eq {
                 using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01064
01065
01066
01067
              template<typename v1>
01068
              struct pos {
01069
                  using type = std::bool_constant<(v1::v > 0)>;
01070
01071
01072
          public:
01076
              template<typename v1, typename v2>
01077
              using add_t = typename add<v1, v2>::type;
01078
              template<typename v1, typename v2>
01082
01083
              using sub t = typename sub<v1, v2>::type;
```

```
01088
              template<typename v1, typename v2>
01089
              using mul_t = typename mul<v1, v2>::type;
01090
01094
              template<typename v1, typename v2>
01095
              using div t = typename div<v1, v2>::type;
01096
01100
              template<typename v1, typename v2>
01101
              using mod_t = typename remainder<v1, v2>::type;
01102
01106
              template<typename v1, typename v2>
01107
              using gt_t = typename gt<v1, v2>::type;
01108
01112
              template<typename v1, typename v2>
01113
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01114
01118
              template<typename v1, typename v2>
              using lt_t = typename lt<v1, v2>::type;
01119
01120
01124
              template<typename v1, typename v2>
01125
              static constexpr bool lt_v = lt_t<v1, v2>::value;
01126
01130
              template<typename v1, typename v2>
01131
              using eq_t = typename eq<v1, v2>::type;
01132
01136
              template<typename v1, typename v2>
01137
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01138
              template<typename v1, typename v2>
using gcd_t = gcd_t<i32, v1, v2>;
01142
01143
01144
01147
              template<typename v1>
01148
              using pos_t = typename pos<v1>::type;
01149
01152
              template < typename v >
              static constexpr bool pos_v = pos_t<v>::value;
01153
01154
          };
01155
01156
          template<int32_t x>
01157
          struct Embed<zpz<x>, i32> {
01158
              template <typename val>
01159
              using type = i32::val<val::v>;
01160
01161 } // namespace aerobus
01162
01163 // polynomial
01164 namespace aerobus {
          // coeffN x^N + ..
01165
01170
          template<typename Ring>
01171
          requires IsEuclideanDomain<Ring>
01172
          struct polynomial {
01173
              static constexpr bool is_field = false;
01174
              static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01175
01179
              template<typename coeffN, typename... coeffs>
01180
              struct val {
                 using ring_type = Ring;
01184
                  using enclosing_type = polynomial<Ring>;
01186
                  static constexpr size_t degree = sizeof...(coeffs);
                  using aN = coeffN;
01188
01190
                  using strip = val<coeffs...>;
                  using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01192
01194
                  static constexpr bool is_zero_v = is_zero_t::value;
01195
01196
               private:
01197
                  template<size_t index, typename E = void>
01198
                  struct coeff_at {};
01199
01200
                  template<size_t index>
                  struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))>> {
01201
01202
                     using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01203
01204
01205
                  template<size t index>
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))» {
01206
01207
                      using type = typename Ring::zero;
01208
01209
               public:
01210
                  template<size_t index>
01213
01214
                  using coeff at t = typename coeff at<index>::type;
                  static std::string to_string() {
01218
01219
                       return string_helper<coeffN, coeffs...>::func();
01220
                  }
01221
01226
                  template<tvpename valueRing>
```

```
static constexpr DEVICE INLINED valueRing eval(const valueRing& x) {
01227
                     return horner_evaluation<valueRing, val>
::template inner<0, degree + 1>
01228
01229
01230
                               ::func(static_cast<valueRing>(0), x);
01231
01232
              };
01233
01236
              template<typename coeffN>
01237
              struct val<coeffN> {
                  using ring_type = Ring;
01239
                  using enclosing_type = polynomial<Ring>;
01241
01243
                  static constexpr size_t degree = 0;
01244
                  using aN = coeffN;
01245
                  using strip = val<coeffN>;
01246
                  using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01247
                  static constexpr bool is zero v = is zero t::value;
01248
01249
01250
                  template<size_t index, typename E = void>
01251
                  struct coeff_at {};
01252
01253
                  template<size_t index>
                  struct coeff_at<index, std::enable_if_t<(index == 0)» {</pre>
01254
01255
                      using type = aN;
01256
                  };
01257
01258
                  template<size_t index>
01259
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01260
                      using type = typename Ring::zero;
01261
01262
01263
                  template<size_t index>
01264
                  using coeff_at_t = typename coeff_at<index>::type;
01265
01266
                  static std::string to_string() {
                      return string_helper<coeffN>::func();
01267
                  }
01268
01269
01270
                  template<typename valueRing>
01271
                  static constexpr DEVICE INLINED valueRing eval(const valueRing& x) {
01272
                       return static_cast<valueRing>(aN::template get<valueRing>());
01273
01274
              }:
01275
01277
              using zero = val<typename Ring::zero>;
01279
              using one = val<typename Ring::one>;
01281
              using X = val<typename Ring::one, typename Ring::zero>;
01282
01283
           private:
01284
             template<typename P, typename E = void>
              struct simplify;
01286
01287
              template <typename P1, typename P2, typename I>
01288
              struct add_low;
01289
01290
              template<typename P1, typename P2>
01291
              struct add {
01292
                  using type = typename simplify<typename add_low<
01293
                  P1,
01294
                  P2.
01295
                  internal::make_index_sequence_reverse<</pre>
                  std::max(P1::degree, P2::degree) + 1
01296
01297
                  »::type>::type;
01298
01299
01300
              template <typename P1, typename P2, typename I>
01301
              struct sub_low;
01302
01303
              template <typename P1, typename P2, typename I>
01304
              struct mul_low;
01305
01306
              template<typename v1, typename v2>
01307
              struct mul {
01308
                      using type = typename mul_low<
01309
                          v1.
01310
01311
                           internal::make_index_sequence_reverse<
01312
                          v1::degree + v2::degree + 1
01313
                          »::type;
01314
              };
01315
01316
              template<typename coeff, size_t deg>
01317
              struct monomial;
01318
01319
              template<typename v, typename E = void>
01320
              struct derive_helper {};
01321
```

```
template<typename v>
               struct derive_helper<v, std::enable_if_t<v::degree == 0» {</pre>
01323
01324
                    using type = zero;
01325
01326
01327
               template<tvpename v>
01328
               struct derive_helper<v, std::enable_if_t<v::degree != 0» {</pre>
01329
                    using type = typename add<
01330
                         typename derive_helper<typename simplify<typename v::strip>::type>::type,
                         typename monomial<
01331
                             typename Ring::template mul_t<</pre>
01332
01333
                                 typename v::aN,
01334
                                  typename Ring::template inject_constant_t<(v::degree)>
01335
01336
                             v::degree - 1
01337
                         >::type
01338
                    >::type;
01339
               };
01340
01341
                template<typename v1, typename v2, typename E = void>
01342
               struct eq_helper {};
01343
               template<typename v1, typename v2>
struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {</pre>
01344
01345
01346
                    using type = std::false_type;
01347
01348
01349
01350
               template<typename v1, typename v2>
01351
               struct eq_helper<v1, v2, std::enable_if_t<
    v1::degree == v2::degree &&</pre>
01352
01353
                    (v1::degree != 0 || v2::degree != 0) &&
01354
                    std::is_same<
01355
                    typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01356
                    std::false_type
01357
                    >::value
01358
01359
               > {
01360
                    using type = std::false_type;
01361
01362
               template<typename v1, typename v2>
struct eq_helper<v1, v2, std::enable_if_t<
    v1::degree == v2::degree &&</pre>
01363
01364
01365
01366
                    (v1::degree != 0 || v2::degree != 0) &&
01367
                    std::is_same<
01368
                    typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01369
                    std::true_type
01370
                    >::value
01371
               » {
01372
                    using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01373
01374
               template<typename v1, typename v2>
struct eq_helper<v1, v2, std::enable_if_t<</pre>
01375
01376
                    v1::degree == v2::degree &&
01377
01378
                    (v1::degree == 0)
01379
               » {
01380
                    using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01381
               };
01382
01383
               template<typename v1, typename v2, typename E = void>
01384
               struct lt_helper {};
01385
01386
               template<typename v1, typename v2>
01387
               struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {</pre>
01388
                    using type = std::true_type;
01389
               };
01390
01391
               template<typename v1, typename v2>
01392
               struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01393
                    using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01394
01395
               template<typename v1, typename v2>
struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
01396
01397
01398
                   using type = std::false_type;
01399
01400
01401
               template<typename v1, typename v2, typename E = void>
01402
               struct gt_helper {};
01403
               template<typename v1, typename v2>
struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
01404
01405
01406
                    using type = std::true_type;
01407
               };
01408
```

```
template<typename v1, typename v2>
               struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01410
01411
                   using type = std::false_type;
01412
01413
               template<typename v1, typename v2>
01414
              struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
01415
01416
                   using type = std::false_type;
01417
01418
               // when high power is zero : strip
01419
01420
               template<tvpename P>
01421
               struct simplify<P, std::enable_if_t<
                  std::is_same<
01422
01423
                   typename Ring::zero,
01424
                   typename P::aN
                   >::value && (P::degree > 0)
01425
01426
               » {
01427
                  using type = typename simplify<typename P::strip>::type;
01428
               } ;
01429
01430
               // otherwise : do nothing
01431
               template<typename P>
01432
               struct simplify<P, std::enable_if_t<
01433
                   !std::is_same<
                   typename Ring::zero,
01434
01435
                   typename P::aN
01436
                   >::value && (P::degree > 0)
01437
               » {
                   using type = P;
01438
01439
               };
01440
01441
               // do not simplify constants
01442
               template<typename P>
01443
               struct simplify<P, std::enable_if_t<P::degree == 0» {</pre>
01444
                   using type = P;
01445
               };
01447
01448
               template<typename P1, typename P2, size_t index>
01449
               struct add_at {
01450
                   using type =
01451
                       typename Ring::template add t<
01452
                           typename P1::template coeff_at_t<index>,
                           typename P2::template coeff_at_t<index>>;
01453
01454
01455
01456
               template<typename P1, typename P2, size_t index>
               using add_at_t = typename add_at<P1, P2, index>::type;
01457
01458
01459
               template<typename P1, typename P2, std::size_t... I>
01460
               struct add_low<P1, P2, std::index_sequence<I...» {
01461
                   using type = val<add_at_t<P1, P2, I>...>;
01462
01463
               // substraction at
01464
               template<typename P1, typename P2, size_t index>
01466
               struct sub_at {
01467
                   using type =
01468
                       typename Ring::template sub_t<</pre>
                           typename P1::template coeff_at_t<index>,
01469
                           typename P2::template coeff_at_t<index>>;
01470
01471
               };
01472
01473
               template<typename P1, typename P2, size_t index>
01474
               using sub_at_t = typename sub_at<P1, P2, index>::type;
01475
              template<typename P1, typename P2, std::size_t... I>
struct sub_low<P1, P2, std::index_sequence<I...» {
    using type = val<sub_at_t<P1, P2, I>...>;
01476
01477
01478
01479
01480
01481
               template<typename P1, typename P2>
01482
               struct sub {
01483
                   using type = typename simplify<typename sub low<
01484
                   P1,
01485
                   P2,
01486
                   internal::make_index_sequence_reverse<</pre>
01487
                   std::max(P1::degree, P2::degree) + 1
01488
                   »::tvpe>::tvpe;
01489
01490
01491
               // multiplication at
01492
               template<typename v1, typename v2, size_t k, size_t index, size_t stop>
01493
               struct mul_at_loop_helper {
                   using type = typename Ring::template add_t<</pre>
01494
                       typename Ring::template mul_t<
01495
```

```
typename v1::template coeff_at_t<index>,
01497
                      typename v2::template coeff_at_t<k - index>
01498
01499
                      typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
01500
01501
              };
01502
01503
              template<typename v1, typename v2, size_t k, size_t stop>
01504
              struct mul_at_loop_helper<v1, v2, k, stop, stop> {
                  using type = typename Ring::template mul_t<
    typename v1::template coeff_at_t<stop>,
01505
01506
                      typename v2::template coeff_at_t<0>>;
01507
01508
              };
01509
01510
              template <typename v1, typename v2, size_t k, typename E = void>
01511
              struct mul_at {};
01512
              01513
01514
01515
                 using type = typename Ring::zero;
01516
01517
             01518
01519
01520
                 using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01521
01522
01523
              template<typename P1, typename P2, size_t index>
01524
              using mul_at_t = typename mul_at<P1, P2, index>::type;
01525
             template<typename P1, typename P2, std::size_t... I>
struct mul_low<P1, P2, std::index_sequence<I...» {</pre>
01526
01527
01528
                 using type = val<mul_at_t<P1, P2, I>...>;
01529
              };
01530
              // division helper
01531
01532
              template< typename A, typename B, typename Q, typename R, typename E = void>
              struct div_helper {};
01534
01535
              template<typename A, typename B, typename Q, typename R>
01536
              struct div_helper<A, B, Q, R, std::enable_if_t<
                  (R::degree < B::degree) ||
01537
                  (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
01538
01539
                 using q_type = Q;
                 using mod_type = R;
01540
01541
                  using gcd_type = B;
01542
01543
             01544
01545
01547
                  !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
               private: // NOLINT
  using rN = typename R::aN;
  using bN = typename B::aN;
01548
01549
01550
01551
                  using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
     B::degree>::type;
01552
                 using rr = typename sub<R, typename mul<pT, B>::type>::type;
                 using qq = typename add<Q, pT>::type;
01553
01554
01555
               public:
                 using q_type = typename div_helper<A, B, qq, rr>::q_type; using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
01556
01558
                 using gcd_type = rr;
01559
01560
01561
              template<typename A, typename B>
01562
              struct div {
01563
                 static_assert (Ring::is_euclidean_domain, "cannot divide in that type of Ring");
                 using q_type = typename div_helper<A, B, zero, A>::q_type;
01564
01565
                 using m_type = typename div_helper<A, B, zero, A>::mod_type;
01566
01567
              template<typename P>
01568
01569
              struct make unit {
01570
                 using type = typename div<P, val<typename P::aN>>::q_type;
01571
01572
01573
              template<typename coeff, size_t deg>
01574
              struct monomial {
01575
                 using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
01576
01577
01578
              template<typename coeff>
01579
              struct monomial<coeff, 0> {
01580
                 using type = val<coeff>;
01581
              };
```

```
01583
              template<typename valueRing, typename P>
01584
              struct horner_evaluation {
01585
                  template<size_t index, size_t stop>
01586
                  struct inner {
                      static constexpr DEVICE INLINED valueRing func (const valueRing& accum, const
01587
      valueRing& x) {
01588
                           constexpr valueRing coeff =
01589
                              static_cast<valueRing>(P::template coeff_at_t<P::degree - index>::template
      get<valueRing>());
01590
                           return horner evaluation<valueRing, P>::template inner<index + 1, stop>::func(x *
     accum + coeff, x);
01591
01592
                  };
01593
01594
                  template<size_t stop>
01595
                  struct inner<stop, stop> {
                      static constexpr DEVICE INLINED valueRing func (const valueRing& accum, const
01596
     valueRing& x) {
01597
                           return accum;
01598
01599
                  };
01600
              };
01601
01602
              template<typename coeff, typename... coeffs>
01603
              struct string_helper {
01604
                   static std::string func() {
                       std::string tail = string_helper<coeffs...>::func();
std::string result = "";
01605
01606
                       if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
01607
01608
                           return tail:
01609
                       } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
01610
                          if (sizeof...(coeffs) == 1) {
01611
                               result += "x";
01612
                           } else {
                               result += "x^" + std::to_string(sizeof...(coeffs));
01613
01614
                           }
01615
                       } else {
01616
                           if (sizeof...(coeffs) == 1) {
01617
                               result += coeff::to_string() + " x";
01618
                           } else {
                               result += coeff::to_string()
01619
                                       + " x^" + std::to_string(sizeof...(coeffs));
01620
01621
                           }
01622
01623
                       if (!tail.empty()) {
    result += " + " + tail;
01624
01625
01626
01627
01628
                       return result;
01629
01630
              } ;
01631
              template<tvpename coeff>
01632
              struct string_helper<coeff>
01633
                  static std::string func() {
01634
01635
                      if (!std::is_same<coeff, typename Ring::zero>::value) {
01636
                          return coeff::to_string();
                       } else {
01637
                           return "";
01638
01639
01640
                  }
01641
              };
01642
           public:
01643
01646
              template<typename P>
              using simplify_t = typename simplify<P>::type;
01647
01648
01652
              template<typename v1, typename v2>
01653
              using add_t = typename add<v1, v2>::type;
01654
01658
              template<typename v1, typename v2>
01659
              using sub_t = typename sub<v1, v2>::type;
01660
01664
              template<typename v1, typename v2>
01665
              using mul_t = typename mul<v1, v2>::type;
01666
01670
              template<typename v1, typename v2>
01671
              using eq_t = typename eq_helper<v1, v2>::type;
01672
01676
              template<typename v1, typename v2>
01677
              using lt_t = typename lt_helper<v1, v2>::type;
01678
01682
              template<typename v1, typename v2> \,
01683
              using gt_t = typename gt_helper<v1, v2>::type;
01684
```

```
template<typename v1, typename v2>
01689
              using div_t = typename div<v1, v2>::q_type;
01690
01694
              template<typename v1, typename v2>
01695
              using mod t = typename div helper<v1, v2, zero, v1>::mod type;
01696
01700
              template<typename coeff, size_t deg>
01701
              using monomial_t = typename monomial<coeff, deg>::type;
01702
01705
              template < typename v >
              using derive_t = typename derive_helper<v>::type;
01706
01707
01710
              template<typename v>
01711
              using pos_t = typename Ring::template pos_t<typename v::aN>;
01712
01715
              template < typename v >
01716
              static constexpr bool pos_v = pos_t < v > :: value;
01717
01721
             template<typename v1, typename v2>
01722
             using gcd_t = std::conditional_t<</pre>
                 Ring::is_euclidean_domain,
01723
01724
                 typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
01725
                 void>:
01726
01730
              template<auto x>
01731
             using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
01732
01736
              template<typename v>
01737
             using inject_ring_t = val<v>;
01738
         };
01739 } // namespace aerobus
01740
01741 // fraction field
01742 namespace aerobus {
01743
         namespace internal {
             template<typename Ring, typename E = void>
01744
01745
              requires IsEuclideanDomain<Ring>
01746
             struct _FractionField {};
01747
01748
              template<typename Ring>
01749
              requires IsEuclideanDomain<Ring>
01750
              01752
                 static constexpr bool is field = true;
                 static constexpr bool is_euclidean_domain = true;
01753
01754
01755
              private:
01756
                 template<typename val1, typename val2, typename E = void>
01757
                 struct to_string_helper {};
01758
01759
                 template<typename val1, typename val2>
01760
                 struct to_string_helper <val1, val2,
01761
                      std::enable_if_t<
01762
                      Ring::template eq_t<
01763
                     val2, typename Ring::one
01764
                     >::value
01765
01766
                 > {
01767
                      static std::string func() {
01768
                         return vall::to_string();
01769
01770
                 };
01771
01772
                  template<typename val1, typename val2>
01773
                  struct to_string_helper<val1, val2,
01774
                      std::enable_if_t<
01775
                      !Ring::template eq_t<
01776
                      val2,
01777
                     typename Ring::one
01778
                      >::value
01779
01780
01781
                      static std::string func() {
                         return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
01782
01783
01784
                 };
01785
01786
               public:
01790
                 template<typename val1, typename val2>
01791
                  struct val {
01793
                     using x = val1;
01795
                     using y = val2;
01797
                     using is_zero_t = typename val1::is_zero_t;
01799
                     static constexpr bool is_zero_v = val1::is_zero_t::value;
01800
                      using ring_type = Ring;
01802
                     using enclosing_type = _FractionField<Ring>;
01803
01804
```

```
static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
01808
01812
                       template<typename valueType>
                       static constexpr DEVICE INLINED valueType get() {
01813
01814
                          return static_cast<valueType>(x::v) / static_cast<valueType>(y::v);
01815
01816
01819
                      static std::string to_string() {
01820
                          return to_string_helper<val1, val2>::func();
01821
01822
01827
                      template<tvpename valueRing>
                      static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
01828
01829
                          return x::eval(v) / y::eval(v);
01830
01831
                  };
01832
01834
                  using zero = val<typename Ring::zero, typename Ring::one>;
01836
                  using one = val<typename Ring::one, typename Ring::one>;
01837
01840
                  template<typename v>
01841
                  using inject_t = val<v, typename Ring::one>;
01842
01845
                  template<auto x>
                  using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
01846
     Ring::one>;
01847
                  template < typename v >
01850
                  using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
01851
01852
01854
                  using ring type = Ring;
01855
01856
01857
                  template<typename v, typename E = void>
01858
                  struct simplify {};
01859
                  // x = 0
01860
01861
                  template<typename v>
01862
                  struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {</pre>
01863
                     using type = typename _FractionField<Ring>::zero;
01864
01865
                  // x != 0
01866
01867
                  template<typename v>
01868
                  struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value» {</pre>
01869
01870
                      using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
                      using newx = typename Ring::template div_t<typename v::x, _gcd>;
01871
                      using newy = typename Ring::template div_t<typename v::y, _gcd>;
01872
01873
01874
                      using posx = std::conditional_t<
01875
                                           !Ring::template pos_v<newy>,
01876
                                           typename Ring::template sub_t<typename Ring::zero, newx>,
01877
                                           newx>;
01878
                      using posy = std::conditional_t<
01879
                                           !Ring::template pos v<newy>,
                                           typename Ring::template sub_t<typename Ring::zero, newy>,
01880
01881
                                           newv>;
01882
                   public:
01883
                      using type = typename _FractionField<Ring>::template val<posx, posy>;
01884
                  }:
01885
01886
               public:
01889
                  template<typename v>
01890
                  using simplify_t = typename simplify<v>::type;
01891
01892
               private:
                  template<typename v1, typename v2> ^{\circ}
01893
01894
                  struct add {
01895
                   private:
01896
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01897
                      using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01898
                      using dividend = typename Ring::template add_t<a, b>;
                      using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01899
01900
                      using g = typename Ring::template gcd_t<dividend, diviser>;
01901
01902
01903
                      using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
     diviser»:
01904
01905
                  template<typename v>
01907
                      using type = std::conditional_t<
01908
01909
                           (\texttt{Ring::template pos\_v<typename v::x> \&\& Ring::template pos\_v<typename v::y>) \ |\ |
01910
                            (!Ring::template pos_v < typename v::x > \&\& !Ring::template pos_v < typename v::y >), \\
01911
                          std::true type,
```

```
01912
                          std::false_type>;
01913
01914
01915
                  template<typename v1, typename v2>
01916
                  struct sub {
01917
                   private:
01918
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01919
                      using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01920
                      using dividend = typename Ring::template sub_t<a, b>;
                      using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01921
                      using g = typename Ring::template gcd_t<dividend, diviser>;
01922
01923
01924
                   public:
                      using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
01925
     diviser»;
01926
01927
01928
                  template<typename v1, typename v2>
01929
                  struct mul {
01930
                   private:
01931
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
01932
                      using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01933
01934
                   public:
01935
                      using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
01936
01937
01938
                  template<typename v1, typename v2, typename E = void>
01939
                  struct div {};
01940
01941
                  template<typename v1, typename v2>
01942
                  struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
     _FractionField<Ring>::zero>::value» {
01943
                   private:
01944
                      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>;
01945
01946
01947
01948
                      using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
01949
                  };
01950
01951
                  template<typename v1, typename v2>
01952
                  struct div<v1. v2. std::enable if t<
01953
                      std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
01954
                      using type = one;
01955
01956
01957
                  template<typename v1, typename v2>
01958
                  struct eq {
01959
                      using type = std::conditional t<
01960
                               std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
01961
                               std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
01962
                          std::true_type,
01963
                          std::false_type>;
01964
                  };
01965
01966
                  template<typename v1, typename v2, typename E = void>
01967
                  struct at:
01968
01969
                  template<typename v1, typename v2>
                  struct gt<v1, v2, std::enable_if_t<
01970
01971
                      (eq<v1, v2>::type::value)
01972
01973
                      using type = std::false_type;
01974
                  } ;
01975
01976
                  template<typename v1, typename v2>
                  struct gt<v1, v2, std::enable_if_t<
01977
                      (!eq<v1, v2>::type::value) &&
01978
01979
                       (!pos<v1>::type::value) && (!pos<v2>::type::value)
01980
01981
                      using type = typename gt<
01982
                          typename sub<zero, v1>::type, typename sub<zero, v2>::type
01983
                      >::type;
01984
                  };
01985
01986
                  template<typename v1, typename v2>
01987
                  struct gt<v1, v2, std::enable_if_t<
01988
                       (!eq<v1, v2>::type::value) &&
                       (pos<v1>::type::value) && (!pos<v2>::type::value)
01989
01990
01991
                      using type = std::true_type;
01992
01993
01994
                  template<typename v1, typename v2>
                  struct gt<v1, v2, std::enable_if_t<
    (!eq<v1, v2>::type::value) &&
01995
01996
```

```
(!pos<v1>::type::value) && (pos<v2>::type::value)
01998
01999
                      using type = std::false_type;
02000
                  };
02001
02002
                  template<tvpename v1, tvpename v2>
                  struct gt<v1, v2, std::enable_if_t<
02003
02004
                       (!eq<v1, v2>::type::value) &&
02005
                       (pos<v1>::type::value) && (pos<v2>::type::value)
02006
02007
                      using type = typename Ring::template gt_t<
02008
                          typename Ring::template mul_t<v1::x, v2::y>,
                          typename Ring::template mul_t<v2::y, v2::x>
02009
02010
02011
                  };
02012
02013
               public:
02018
                  template<typename v1, typename v2>
02019
                  using add_t = typename add<v1, v2>::type;
02020
02025
                  template<typename v1, typename v2>
02026
                  using mod_t = zero;
02027
02032
                  template<typename v1, typename v2>
02033
                  using gcd_t = v1;
02034
02038
                  template<typename v1, typename v2>
02039
                  using sub_t = typename sub<v1, v2>::type;
02040
02044
                  template<typename v1, typename v2>
02045
                  using mul t = typename mul<v1, v2>::type;
02046
02050
                  template<typename v1, typename v2>
02051
                  using div_t = typename div<v1, v2>::type;
02052
02056
                  template<typename v1, typename v2>
02057
                  using eq_t = typename eq<v1, v2>::type;
02058
02062
                  template<typename v1, typename v2>
02063
                  static constexpr bool eq_v = eq<v1, v2>::type::value;
02064
02068
                  template<typename v1, typename v2>
02069
                  using gt_t = typename gt<v1, v2>::type;
02070
02074
                  template<typename v1, typename v2>
02075
                  static constexpr bool gt_v = gt<v1, v2>::type::value;
02076
02079
                  template<typename v1>
02080
                  using pos_t = typename pos<v1>::type;
02081
02084
                  template<typename v>
02085
                  static constexpr bool pos_v = pos_t<v>::value;
02086
              };
02087
02088
              template<typename Ring, typename E = void>
02089
              requires IsEuclideanDomain<Ring>
02090
              struct FractionFieldImpl {};
02091
02092
              // fraction field of a field is the field itself
02093
              template<typename Field>
              requires IsEuclideanDomain<Field>
02094
02095
              struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {</pre>
02096
                  using type = Field;
02097
                  template<typename v>
02098
                  using inject_t = v;
02099
              };
02100
              // fraction field of a ring is the actual fraction field
02101
02102
              template<typename Ring>
02103
              requires IsEuclideanDomain<Ring>
02104
              struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {</pre>
02105
                  using type = _FractionField<Ring>;
02106
          } // namespace internal
02107
02108
02112
          template<typename Ring>
02113
          requires IsEuclideanDomain<Ring>
02114
          using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02115
02116
          template<tvpename Ring>
          struct Embed<Ring, FractionField<Ring» {</pre>
02117
02118
              template<typename v>
02119
              using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02120
02121 }
         // namespace aerobus
02122
02123
```

```
02124 // short names for common types
02125 namespace aerobus {
02128
          using q32 = FractionField<i32>;
02129
02132
          using fpg32 = FractionField<polynomial<g32>>;
02133
02136
          using q64 = FractionField<i64>;
02137
02139
          using pi64 = polynomial<i64>;
02140
          using pg64 = polynomial<g64>;
02142
02143
02145
          using fpq64 = FractionField<polynomial<q64>>;
02146
02151
          template<typename Ring, typename v1, typename v2>
02152
          using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02153
02160
          template<typename v>
02161
          using embed_int_poly_in_fractions_t =
02162
                  typename Embed<
02163
                      polynomial<typename v::ring_type>,
02164
                       polynomial<FractionField<typename v::ring_type>»::template type<v>;
02165
02169
          template<int64_t p, int64_t q>
using make_q64_t = typename q64::template simplify_t
02170
02171
                      typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02172
          template<int32_t p, int32_t q>
using make_q32_t = typename q32::template simplify_t<</pre>
02176
02177
                       typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>»;
02178
02179
02184
          template<typename Ring, typename v1, typename v2>
02185
          using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02190
          template<typename Ring, typename v1, typename v2>
02191
          using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02192
02193
          template<>
02194
          struct Embed<q32, q64> {
02195
              template<typename v>
02196
              using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02197
02198
02199
          template<typename Small, typename Large>
02200
          struct Embed<polynomial<Small>, polynomial<Large» {</pre>
          private:
02201
02202
              template<typename v, typename i>
02203
              struct at_low;
02204
02205
              template<typename v, size_t i>
02206
              struct at_index {
02207
                  using type = typename Embed<Small, Large>::template
      type<typename v::template coeff_at_t<i>>;
02208
02209
02210
              template<typename v, size_t... Is>
              struct at_low<v, std::index_sequence<Is...» {
   using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
02211
02212
02213
              };
02214
           public:
02215
02216
              template<typename v>
              using type = typename at_low<v, typename internal::make_index_sequence_reverse<v::degree +</pre>
02217
     1»::type;
02218
02219
          template<typename Ring, auto... xs>
02223
02224
          using make_int_polynomial_t = typename polynomial<Ring>::template val<</pre>
                  typename Ring::template inject_constant_t<xs>...>;
02225
02226
02230
          template<typename Ring, auto... xs>
02231
          using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<
02232
                  typename FractionField<Ring>::template inject_constant_t<xs>...>;
02233 } // namespace aerobus
02234
02235 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02236 namespace aerobus {
02237
        namespace internal {
02238
             template<typename T, size_t x, typename E = void>
02239
              struct factorial {};
02240
02241
              template<typename T, size_t x>
02242
              struct factorial<T, x, std::enable_if_t<(x > 0)» {
02243
              private:
02244
                  template<typename, size_t, typename>
02245
                   friend struct factorial;
02246
              public:
02247
                  using type = typename T::template mul t<typename T::template val<x>, typename factorial<T.
```

```
x - 1>::type>;
02248
                  static constexpr typename T::inner_type value = type::template get<typename</pre>
     T::inner_type>();
02249
             } ;
02250
02251
              template<typename T>
             struct factorial<T, 0> {
02253
              public:
                using type = typename T::one;
02254
02255
                 static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02256
              };
          } // namespace internal
02257
02258
02262
          template<typename T, size_t i>
02263
          using factorial_t = typename internal::factorial<T, i>::type;
02264
02268
          template<typename T, size t i>
          inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02269
02270
02271
          namespace internal {
02272
              template<typename T, size_t k, size_t n, typename E = void>
02273
              struct combination_helper {};
02274
02275
              template<typename T, size_t k, size_t n>
02276
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
02277
                  using type = typename FractionField<T>::template mul_t<</pre>
02278
                      typename combination_helper<T, k - 1, n - 1>::type,
02279
                      makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02280
              };
02281
02282
              template<typename T, size_t k, size_t n>
02283
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)» {
02284
                  using type = typename combination_helper<T, n - k, n>::type;
02285
02286
02287
              template<typename T, size_t n>
              struct combination_helper<T, 0, n> {
02289
                 using type = typename FractionField<T>::one;
02290
02291
02292
              template<typename T, size_t k, size_t n>
02293
              struct combination {
02294
                  using type = typename internal::combination_helper<T, k, n>::type::x;
02295
                  static constexpr typename T::inner_type value =
02296
                              internal::combination_helper<T, k, n>::type::template get<typename</pre>
     T::inner_type>();
02297
          };
} // namespace internal
02298
02299
02302
          template<typename T, size_t k, size_t n>
02303
          using combination_t = typename internal::combination<T, k, n>::type;
02304
02309
          template<typename T, size_t k, size_t n> 
         inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02310
02311
02312
          namespace internal {
02313
             template<typename T, size_t m>
02314
              struct bernoulli;
02315
              template<typename T, typename accum, size_t k, size_t m>
02316
02317
              struct bernoulli helper {
02318
                  using type = typename bernoulli_helper<
02319
02320
                      addfractions_t<T,
02321
                          accum,
02322
                          mulfractions_t<T,</pre>
02323
                              makefraction t<T.
02324
                                  combination_t<T, k, m + 1>,
02325
                                  typename T::one>,
02326
                               typename bernoulli<T, k>::type
02327
                      >,
k + 1,
02328
02329
02330
                      m>::type;
02331
              };
02332
02333
              template<typename T, typename accum, size_t m>
02334
              struct bernoulli_helper<T, accum, m, m> {
02335
                 using type = accum;
02336
02337
02338
02339
02340
              template<typename T, size_t m>
02341
              struct bernoulli {
02342
                  using type = typename FractionField<T>::template mul_t<</pre>
```

```
02343
                       typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02344
                       makefraction t<T,
02345
                       typename T::template val<static_cast<typename T::inner_type>(-1)>,
02346
                       typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02347
02348
                  >;
02349
02350
                  template<typename floatType>
02351
                  static constexpr floatType value = type::template get<floatType>();
02352
              } ;
02353
02354
              template<typename T>
02355
              struct bernoulli<T, 0> {
                  using type = typename FractionField<T>::one;
02356
02357
02358
                  template<typename floatType>
02359
                  static constexpr floatType value = type::template get<floatType>();
02360
              };
02361
          } // namespace internal
02362
02366
          template<typename T, size_t n>
02367
          using bernoulli_t = typename internal::bernoulli<T, n>::type;
02368
          template<typename FloatType, typename T, size_t n >
inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02373
02374
02375
02376
          // bell numbers
02377
          namespace internal {
              template<typename T, size_t n, typename E = void>
02378
02379
              struct bell_helper;
02380
02381
              template <typename T, size_t n>
02382
              struct bell_helper<T, n, std::enable_if_t<(n > 1)» {
                  template<typename accum, size_t i, size_t stop>
02383
02384
                   struct sum_helper {
                   private:
02385
02386
                      using left = typename T::template mul_t<</pre>
                                   combination_t<T, i, n-1>,
02387
02388
                                    typename bell_helper<T, i>::type>;
02389
                      using new_accum = typename T::template add_t<accum, left>;
                   public:
02390
02391
                       using type = typename sum_helper<new_accum, i+1, stop>::type;
02392
                  }:
02393
02394
                   template<typename accum, size_t stop>
02395
                   struct sum_helper<accum, stop, stop> {
02396
                      using type = accum;
02397
                  };
02398
02399
                  using type = typename sum helper<typename T::zero, 0, n>::type;
02400
              };
02401
02402
              template<typename T>
02403
              struct bell_helper<T, 0> {
                  using type = typename T::one;
02404
02405
              };
02406
02407
              template<typename T>
02408
              struct bell_helper<T, 1> {
02409
                  using type = typename T::one;
02410
02411
          } // namespace internal
02412
02416
          template<typename T, size_t n>
02417
          using bell_t = typename internal::bell_helper<T, n>::type;
02418
          template<typename T, size_t n>
static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
02422
02423
02424
02425
          namespace internal {
02426
              template<typename T, int k, typename E = void>
02427
              struct alternate {};
02428
02429
              template<typename T, int k>
              struct alternate<T, k, std::enable_if_t<k % 2 == 0» {</pre>
02430
                  using type = typename T::one;
02431
02432
                   static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
02433
              };
02434
02435
              template<typename T, int k>
              struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
02436
                  using type = typename T::template sub_t<typename T::zero, typename T::one>;
02437
02438
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02439
          };
} // namespace internal
02440
```

```
02441
02444
          template<typename T, int k>
02445
          using alternate_t = typename internal::alternate<T, k>::type;
02446
02447
          namespace internal {
              template<typename T, int n, int k, typename E = void>
02448
02449
              struct stirling_helper {};
02450
02451
              template<typename T>
              struct stirling_helper<T, 0, 0> {
02452
02453
                   using type = typename T::one;
02454
02455
02456
              template<typename T, int n>
02457
              struct stirling_helper<T, n, 0, std::enable_if_t<(n > 0)» {
02458
                 using type = typename T::zero;
02459
              };
02460
02461
              template<typename T, int n>
02462
              struct stirling_helper<T, 0, n, std::enable_if_t<(n > 0)» {
02463
                 using type = typename T::zero;
02464
02465
02466
              template<typename T, int n, int k>
02467
              struct stirling_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)» {
                 using type = typename T::template sub_t<
02468
02469
                                    typename stirling_helper<T, n-1, k-1>::type,
02470
                                    typename T::template mul_t<</pre>
02471
                                        typename T::template inject_constant_t<n-1>,
                                        typename stirling_helper<T, n-1, k>::type
02472
02473
02474
02475
          } // namespace internal
02476
02481
          template<typename T, int n, int k>
          using stirling_signed_t = typename internal::stirling_helper<T, n, k>::type;
02482
02483
          template<typename T, int n, int k>
02489
          using stirling_unsigned_t = abs_t<typename internal::stirling_helper<T, n, k>::type>;
02490
02495
          template<typename T, int n, int k>
          static constexpr typename T::inner_type stirling_signed_v = stirling_signed_t<T, n, k>::v;
02496
02497
02498
02503
          template<typename T, int n, int k>
02504
          static constexpr typename T::inner_type stirling_unsigned_v = stirling_unsigned_t<T, n, k>::v;
02505
02508
          template<typename T, size_t k>
          inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
02509
02510
02511
          namespace internal {
02512
              template<typename T>
02513
               struct pow_scalar {
02514
                   template<size_t p>
                   template=12e_t ps
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) :
02515
02516
                       x * func<p/2>(x) * func<p/2>(x);
02517
02518
                   }
02519
02520
02521
              template<typename T, typename p, size_t n, typename E = void>
02522
              requires IsEuclideanDomain<T>
              struct pow;
02524
02525
              template<typename T, typename p, size_t n>
              struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)» {
02526
02527
                  using type = typename T::template mul_t<
                       typename pow<T, p, n/2>::type, typename pow<T, p, n/2>::type
02528
02529
02530
02531
              };
02532
              template<typename T, typename p, size_t n>
struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)» {
    using type = typename T::template mul_t</pre>
02533
02534
02535
02536
                      p,
02537
                       typename T::template mul_t<
                           typename pow<T, p, n/2>::type, typename pow<T, p, n/2>::type
02538
02539
02540
02541
                   >;
02542
              };
02543
02544
              template<typename T, typename p, size_t n>
02545
              } // namespace internal
02546
02547
```

```
template<typename T, typename p, size_t n>
02553
          using pow_t = typename internal::pow<T, p, n>::type;
02554
02559
          template<typename T, typename p, size_t n>
          static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
02560
02561
02562
          template<typename T, size_t p>
02563
          static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
      internal::pow_scalar<T>::template func(x); }
02564
02565
          namespace internal {
02566
              template<typename, template<typename, size_t> typename, class>
02567
               struct make taylor impl;
02568
02569
               template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
              struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...» {
   using type = typename polynomial<FractionField<T>::template val<typename coeff_at<T,</pre>
02570
02571
      Is>::type...>;
02572
              };
02573
02574
02579
          template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
02580
          using taylor = typename internal::make_taylor_impl<</pre>
02581
02582
               coeff_at,
02583
              internal::make_index_sequence_reverse<deg + 1>>::type;
02584
02585
          namespace internal {
02586
               template<typename T, size_t i>
02587
               struct exp_coeff {
02588
                   using type = makefraction t<T, typename T::one, factorial t<T, i>>;
02589
02590
02591
               template<typename T, size_t i, typename E = void>
02592
               struct sin_coeff_helper {};
02593
02594
               template<typename T, size_t i>
02595
               struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
02596
                  using type = typename FractionField<T>::zero;
02597
02598
02599
               template<typename T, size_t i>
02600
               struct sin coeff helper<T, i, std::enable if t<(i & 1) == 1» {
02601
                   using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02602
02603
02604
               template<typename T, size_t i>
02605
               struct sin_coeff {
                  using type = typename sin_coeff_helper<T, i>::type;
02606
02607
02608
02609
               template<typename T, size_t i, typename E = void>
02610
               struct sh_coeff_helper {};
02611
               template<typename T, size_t i>
02612
               struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
02613
02614
02615
02616
02617
               template<typename T, size_t i>
               struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02618
02619
                   using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02620
02621
02622
               template<typename T, size_t i>
02623
               struct sh_coeff {
02624
                   using type = typename sh_coeff_helper<T, i>::type;
02625
02626
               template<typename T, size_t i, typename E = void>
02628
               struct cos_coeff_helper {};
02629
02630
               template<typename T, size_t i>
               struct cos_coeff_helperTT, i, std::enable_if_t<(i & 1) == 1» {
    using type = typename FractionField<T>::zero;
02631
02632
02633
02634
               template<typename T, size_t i>
02635
               struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
02636
                  using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02637
02638
02639
02640
               template<typename T, size_t i>
02641
               struct cos_coeff {
02642
                   using type = typename cos_coeff_helper<T, i>::type;
02643
02644
```

```
template<typename T, size_t i, typename E = void>
               struct cosh_coeff_helper {};
02646
02647
02648
               template<typename T, size t i>
               struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
    using type = typename FractionField<T>::zero;
02649
02650
02651
02652
02653
               template<typename T, size_t i>
02654
               struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0\times {
                   using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02655
02656
02657
02658
               template<typename T, size_t i>
02659
               struct cosh_coeff {
02660
                  using type = typename cosh_coeff_helper<T, i>::type;
02661
               };
02662
02663
               template<typename T, size_t i>
02664
               struct geom_coeff { using type = typename FractionField<T>::one; };
02665
02666
02667
               template<typename T, size_t i, typename E = void>
02668
               struct atan coeff helper;
02669
02670
               template<typename T, size_t i>
02671
               struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
02672
                   using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;;
02673
02674
02675
               template<typename T, size_t i>
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
02676
02677
02678
               };
02679
               template<typename T, size_t i>
02680
02681
               struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
02682
02683
               template<typename T, size_t i, typename E = void>
02684
               struct asin_coeff_helper;
02685
02686
               template<typename T, size t i>
               struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
02687
02688
                   using type = makefraction_t<T,
                       factorial_t<T, i - 1>,
02689
02690
                        typename T::template mul_t<
02691
                            typename T::template val<i>,
02692
                            T::template mul_t<
                                pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
02693
                                pow<T, factorial_t<T, i / 2>, 2
02694
02695
02696
02697
02698
              };
02699
02700
               template<typename T, size t i>
02701
               struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
                   using type = typename FractionField<T>::zero;
02702
02703
02704
               template<typename T, size_t i>
02705
               struct asin_coeff {
02706
02707
                  using type = typename asin_coeff_helper<T, i>::type;
02708
02709
02710
               template<typename T, size_t i>
02711
               struct lnp1_coeff {
                  using type = makefraction_t<T,
02712
                       alternate_t<T, i + 1>,
02713
                       typename T::template val<i>;;
02714
02715
              };
02716
02717
               template<typename T>
               struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
02718
02719
02720
               template<typename T, size_t i, typename E = void>
02721
               struct asinh_coeff_helper;
02722
02723
               template<typename T, size_t i>
               struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
02724
                  using type = makefraction_t<T,</pre>
02725
                       typename T::template mul_t<
                           alternate_t<T, i / 2>,
factorial_t<T, i - 1>
02727
02728
02729
02730
                       typename T::template mul_t<
02731
                            typename T::template mul t<
```

```
typename T::template val<i>,
02733
                               pow_t<T, factorial_t<T, i / 2>, 2>
02734
02735
                           pow_t<T, typename T::template inject_constant_t<4>, i / 2>
02736
02737
                  >;
02738
              };
02739
02740
               template<typename T, size_t i>
02741
              struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
02742
                  using type = typename FractionField<T>::zero;
02743
02744
02745
              template<typename T, size_t i>
02746
              struct asinh_coeff {
02747
                  using type = typename asinh_coeff_helper<T, i>::type;
02748
              };
02749
              template<typename T, size_t i, typename E = void>
02751
              struct atanh_coeff_helper;
02752
02753
               template<typename T, size_t i>
              struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
    // 1/i</pre>
02754
02755
02756
                   using type = typename FractionField<T>:: template val<
02757
                      typename T::one,
02758
                       typename T::template inject_constant_t<i>;;
02759
02760
              template<typename T, size_t i>
02761
02762
              struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
02763
                  using type = typename FractionField<T>::zero;
02764
02765
              template<typename T, size_t i>
struct atanh_coeff {
02766
02767
02768
                  using type = typename atanh_coeff_helper<T, i>::type;
02769
02770
02771
              template<typename T, size_t i, typename E = void>
02772
              struct tan_coeff_helper;
02773
02774
              template<typename T, size_t i>
02775
              struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
02776
                  using type = typename FractionField<T>::zero;
02777
02778
02779
              template<typename T, size_t i>
              struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
02780
02781
              private:
                   // 4^((i+1)/2)
02783
                   using _4p = typename FractionField<T>::template inject_t<</pre>
02784
                       pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2\ast;
02785
                   // 4^((i+1)/2) - 1
                  using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename</pre>
02786
      FractionField<T>::one>;
02787
                  // (-1)^((i-1)/2)
02788
                   using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
02789
                   using dividend = typename FractionField<T>::template mul_t<</pre>
02790
                       altp,
                       FractionField<T>::template mul_t<</pre>
02791
02792
                       _4p,
02793
                       FractionField<T>::template mul_t<
02794
                       _4pm1,
02795
                       bernoulli_t<T, (i + 1)>
02796
02797
02798
                  >;
02799
              public:
                  using type = typename FractionField<T>::template div_t<dividend,</pre>
02801
                       typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02802
02803
02804
              template<typename T, size_t i>
02805
              struct tan coeff {
02806
                  using type = typename tan_coeff_helper<T, i>::type;
02807
02808
02809
              template<typename T, size_t i, typename E = void>
02810
              struct tanh_coeff_helper;
02811
02812
               template<typename T, size_t i>
02813
              struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {</pre>
02814
                   using type = typename FractionField<T>::zero;
02815
02816
02817
              template<typename T, size t i>
```

```
struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
02819
02820
                  using _4p = typename FractionField<T>::template inject_t<</pre>
                  pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename</pre>
02821
02822
     FractionField<T>::one>;
02823
                  using dividend =
02824
                       typename FractionField<T>::template mul_t<</pre>
02825
02826
                           typename FractionField<T>::template mul_t<</pre>
02827
                                _4pm1,
                               bernoulli_t<T, (i + 1) >>::type;
02828
02829
              public:
02830
                  using type = typename FractionField<T>::template div_t<dividend,</pre>
02831
                       FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02832
02833
02834
              template<typename T, size_t i>
              struct tanh_coeff {
02835
02836
                  using type = typename tanh_coeff_helper<T, i>::type;
02837
02838
          } // namespace internal
02839
02843
          template<typename Integers, size_t deg>
02844
          using exp = taylor<Integers, internal::exp_coeff, deg>;
02845
02849
          template<typename Integers, size_t deg>
02850
          using expm1 = typename polynomial<FractionField<Integers>>::template sub_t
02851
              exp<Integers, deg>,
              typename polynomial<FractionField<Integers>>::one>;
02852
02853
02857
          template<typename Integers, size_t deg>
02858
          using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
02859
02863
          template<typename Integers, size_t deg>
          using atan = taylor<Integers, internal::atan_coeff, deg>;
02864
02865
02869
          template<typename Integers, size_t deg>
02870
          using sin = taylor<Integers, internal::sin_coeff, deg>;
02871
02875
          template<typename Integers, size_t deg>
02876
          using sinh = taylor<Integers, internal::sh_coeff, deg>;
02877
02882
          template<typename Integers, size_t deg>
          using cosh = taylor<Integers, internal::cosh_coeff, deg>;
02883
02884
02889
          template<typename Integers, size_t deg>
02890
          using cos = taylor<Integers, internal::cos_coeff, deg>;
02891
02896
          template<typename Integers, size_t deg>
02897
          using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
02898
02903
          template<typename Integers, size_t deg>
02904
          using asin = taylor<Integers, internal::asin_coeff, deg>;
02905
02910
          template<typename Integers, size_t deg>
02911
          using asinh = taylor<Integers, internal::asinh_coeff, deg>;
02912
02917
          template<typename Integers, size_t deg>
02918
          using atanh = taylor<Integers, internal::atanh_coeff, deg>;
02919
02924
          template<typename Integers, size_t deg>
02925
          using tan = taylor<Integers, internal::tan_coeff, deg>;
02926
          template<typename Integers, size_t deg>
02931
02932
          using tanh = taylor<Integers, internal::tanh_coeff, deg>;
         // namespace aerobus
02933 }
02934
02935 // continued fractions
02936 namespace aerobus {
02945
          template<int64_t... values>
02946
          struct ContinuedFraction {};
02947
02950
          template<int64 t a0>
02951
          struct ContinuedFraction<a0> {
02953
              using type = typename q64::template inject_constant_t<a0>;
              static constexpr double val = static_cast<double>(a0);
02955
02956
02957
02961
          template<int64 t a0, int64 t... rest>
          struct ContinuedFraction<a0, rest...> {
02962
              using type = q64::template add_t<
02964
02965
                       typename q64::template inject_constant_t<a0>,
02966
                       typename q64::template div_t<
02967
                           typename q64::one,
                           typename ContinuedFraction<rest...>::type
02968
02969
                       »;
```

```
static constexpr double val = type::template get<double>();
02972
02973
02978
                 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>;
02981
                 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>;
02983
                 using SQRT2_fraction
          02985
                using SORT3 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
02986 } // namespace aerobus
02987
02988 // known polynomials
02989 namespace aerobus {
                 // CChebyshev
02990
02991
                 namespace internal {
02992
                       template<int kind, size_t deg, typename I>
02993
                        struct chebyshev_helper {
02994
                               using type = typename polynomial<I>::template sub_t<
02995
                                      typename polynomial<I>::template mul_t<</pre>
02996
                                            typename polynomial<I>::template mul_t<</pre>
02997
                                                    typename polynomial<I>::template inject_constant_t<2>,
02998
                                                    typename polynomial<I>::X>,
02999
                                             typename chebyshev_helper<kind, deg - 1, I>::type
03000
03001
                                      typename chebyshev_helper<kind, deg - 2, I>::type
03002
03003
                        };
03004
03005
                        template<typename I>
03006
                        struct chebyshev_helper<1, 0, I> {
03007
                              using type = typename polynomial<I>::one;
03008
03009
03010
                        template<typename I>
03011
                        struct chebyshev_helper<1, 1, I> {
03012
                              using type = typename polynomial<I>::X;
03013
03014
03015
                        template<typename I>
                        struct chebyshev_helper<2, 0, I> {
    using type = typename polynomial<I>::one;
03016
03017
03018
03019
03020
                        template<typename I>
03021
                        struct chebyshev_helper<2, 1, I> \{
                              using type = typename polynomial<I>::template mul_t<</pre>
03022
03023
                                      typename polynomial<I>::template inject_constant_t<2>,
03024
                                      typename polynomial<I>::X>;
03025
03026
                 } // namespace internal
03027
                 // Laguerre
03028
03029
                 namespace internal {
03030
                        template<size_t deg, typename I>
03031
                        struct laguerre_helper
03032
                               using Q = FractionField<I>;
03033
                               using PQ = polynomial<Q>;
03034
03035
                          private:
03036
                               // Lk = (1 / k) * ((2 * k - 1 - x) * 1km1 - (k - 2)Lkm2)
03037
                               using lnm2 = typename laguerre_helper<deg - 2, I>::type;
03038
                               using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03039
                               // -x + 2k-1
03040
                               using p = typename PQ::template val<
03041
                                      typename Q::template inject_constant_t<-1>,
                                      typename Q::template inject_constant_t<2 * deg - 1»;
03042
                                // 1/n
03043
03044
                               using factor = typename PQ::template inject_ring_t<
03045
                                      typename Q::template val<typename I::one, typename I::template
          inject_constant_t<deg>>;
03046
03047
                          public:
03048
                              using type = typename PQ::template mul_t <
03049
                                      factor,
03050
                                      typename PQ::template sub_t<
03051
                                             typename PQ::template mul_t<
03052
                                                   p,
03053
                                                   lnm1
03054
03055
                                             typename PQ::template mul_t<
03056
                                                    typename PQ::template inject_constant_t<deg-1>,
03057
                                                    lnm2
03058
03059
```

```
03060
                  >;
03061
03062
03063
              template<typename I>
03064
              struct laguerre_helper<0, I> {
                 using type = typename polynomial<FractionField<I»::one;
03065
03066
03067
03068
              template<typename I>
03069
              struct laguerre_helper<1, I> {
03070
              private:
03071
                 using PQ = polynomial<FractionField<I>;
03072
               public:
03073
                 using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
03074
03075
          } // namespace internal
03076
03077
          // Bernstein
03078
          namespace internal {
03079
              template<size_t i, size_t m, typename I, typename E = void>
03080
              struct bernstein_helper {};
03081
03082
              template<typename I>
              struct bernstein_helper<0, 0, I> {
03083
03084
                 using type = typename polynomial<I>::one;
03085
03086
03087
              template<size_t i, size_t m, typename I>
              03088
03089
03090
               private:
03091
                  using P = polynomial<I>;
03092
               public:
03093
                 using type = typename P::template mul_t<</pre>
03094
                          typename P::template sub_t<typename P::one, typename P::X>,
03095
                          typename bernstein_helper<i, m-1, I>::type>;
03096
              };
03097
03098
              template<size_t i, size_t m, typename I>
              struct bernstein_helper<i, m, I, std::enable_if_t<
(m > 0) && (i == m)» {
03099
03100
0.31.01
               private:
                 using P = polynomial<I>;
03102
               public:
03103
03104
                 using type = typename P::template mul_t<
03105
                          typename P::X,
03106
                          typename bernstein_helper<i-1, m-1, I>::type>;
03107
              };
03108
03109
              template<size_t i, size_t m, typename I>
              struct bernstein_helper<i, m, I, std::enable_if_t<
03110
03111
                          (m > 0) && (i > 0) && (i < m)» {
               private:
03112
03113
                  using P = polynomial<I>;
               public:
03114
03115
                  using type = typename P::template add t<
03116
                          typename P::template mul_t<
03117
                              typename P::template sub_t<typename P::one, typename P::X>,
03118
                              typename bernstein_helper<i, m-1, I>::type>,
03119
                          typename P::template mul_t<
03120
                              typename P::X,
03121
                              typename bernstein_helper<i-1, m-1, I>::type»;
03122
03123
          } // namespace internal
03124
03125
          namespace known_polynomials {
             enum hermite_kind {
   probabilist,
03127
03129
03131
                  physicist
03132
              };
03133
         }
03134
          // hermite
03135
          namespace internal {
03136
              template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03137
03138
              struct hermite_helper {};
03139
03140
              template<size_t deg, typename I>
03141
              struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
0.3142
              private:
                 using hnm1 = typename hermite_helper<deg - 1,
03143
      known_polynomials::hermite_kind::probabilist, I>::type;
                 using hnm2 = typename hermite_helper<deg - 2,
      known_polynomials::hermite_kind::probabilist, I>::type;
03145
               public:
03146
03147
                  using type = typename polynomial<I>::template sub t<
```

```
typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnm1>,
                       typename polynomial<I>::template mul_t<
03149
03150
                           typename polynomial<I>::template inject_constant_t<deg - 1>,
03151
                          hnm2
03152
03153
                  >;
03154
              };
03155
03156
              template<size_t deg, typename I>
0.3157
              struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
               private:
03158
03159
                  using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
      I>::tvpe;
03160
                  using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
     I>::type;
03161
03162
               public:
03163
                  using type = typename polynomial<I>::template sub_t<
                       // 2X Hn-1
03164
03165
                       typename polynomial<I>::template mul_t<
03166
                           typename pi64::val<typename I::template inject_constant_t<2>,
                           typename I::zero>, hnm1>,
03167
0.3168
03169
                      typename polynomial<I>::template mul_t<</pre>
03170
                           typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03171
03172
03173
03174
              };
03175
03176
              template<tvpename I>
03177
              struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03178
                  using type = typename polynomial<I>::one;
03179
              };
0.3180
03181
              template<typename I>
03182
              struct hermite helper<1, known polynomials::hermite kind::probabilist, I> {
03183
                  using type = typename polynomial<I>::X;
03184
03185
03186
              template<typename I>
              struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03187
03188
                  using type = typename pi64::one;
03189
03190
03191
              template<typename I>
03192
              struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03193
                  // 2X
03194
                  using type = typename polynomial<I>::template val<
                      typename I::template inject_constant_t<2>,
03195
03196
                      typename I::zero>;
03197
03198
          } // namespace internal
03199
          // legendre
03200
03201
          namespace internal {
03202
              template<size_t n, typename I>
03203
              struct legendre_helper {
03204
               private:
03205
                  using Q = FractionField<I>;
                  using PQ = polynomial<Q>;
03206
03207
                  // 1/n constant
                  // (2n-1)/n X
03208
03209
                  using fact_left = typename PQ::template monomial_t<</pre>
03210
                       makefraction_t<I,
03211
                          typename I::template inject_constant_t<2*n-1>,
03212
                           typename I::template inject_constant_t<n>
03213
                      >,
                  1>;
03214
                  // (n-1) / n
03215
03216
                  using fact_right = typename PQ::template val<
03217
                       makefraction_t<I,
03218
                          typename I::template inject_constant_t<n-1>,
03219
                           typename I::template inject_constant_t<n>>;
03220
03221
               public:
03222
                  using type = PQ::template sub_t<
03223
                           typename PQ::template mul_t<
03224
                               fact left.
                               typename legendre_helper<n-1, I>::type
03225
03226
03227
                           typename PQ::template mul_t<
03228
                               fact_right,
03229
                               typename legendre_helper<n-2, I>::type
03230
03231
                      >;
03232
              };
```

```
03233
03234
                          template<typename I>
03235
                          struct legendre_helper<0, I> {
03236
                               using type = typename polynomial<FractionField<I>::one;
03237
03238
03239
                          template<typename I>
03240
                          struct legendre_helper<1, I> {
03241
                               using type = typename polynomial<FractionField<I»::X;
03242
03243
                  } // namespace internal
03244
03245
                  // bernoulli polynomials
03246
                  namespace internal {
03247
                          template<size_t n>
03248
                          struct bernoulli_coeff {
03249
                                 template<typename T, size_t i>
                                 struct inner {
03250
03251
                                  private:
03252
                                        using F = FractionField<T>;
03253
                                   public:
03254
                                        using type = typename F::template mul_t<</pre>
                                                 typename F::template inject_ring_t<combination_t<T, i, n»,
03255
03256
                                                bernoulli_t<T, n-i>
03257
03258
                                };
03259
03260
                 } // namespace internal
03261
03263
                  namespace known_polynomials {
                          template <size_t deg, typename I = aerobus::i64>
03271
03272
                          using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
03273
                          template <size_t deg, typename I = aerobus::i64>
03283
03284
                          using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
03285
                         template <size_t deg, typename I = aerobus::i64>
using laguerre = typename internal::laguerre_helper<deg, I>::type;
03295
03296
03297
03304
                          template <size_t deg, typename I = aerobus::i64>
03305
                         using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
          I>::tvpe;
03306
03313
                          template <size_t deg, typename I = aerobus::i64>
03314
                          using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
03315
03326
                          template<size_t i, size_t m, typename I = aerobus::i64>
03327
                          using bernstein = typename internal::bernstein_helper<i, m, I>::type;
03328
03338
                          template<size t deg, typename I = aerobus::i64>
03339
                          using legendre = typename internal::legendre_helper<deg, I>::type;
03340
03350
                          template<size_t deg, typename I = aerobus::i64>
                          using bernoulli = taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
03351
                       // namespace known_polynomials
03352
03353 } // namespace aerobus
03355
03356 #ifdef AEROBUS_CONWAY_IMPORTS
03357
03358 // conway polynomials
03359 namespace aerobus {
03363
                 template<int p, int n>
                  struct ConwayPolynomial {};
03364
03365
03366 #ifndef DO_NOT_DOCUMENT
             #define ZPZV ZPZ::template val
03367
                   #define POLYV aerobus::polynomial<ZPZ>::template val
03368
          template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
03369
03370
                  template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
          POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
                  template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
03371
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
03372
                 template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
03373
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT
                 template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
03374
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2), ZPZV<1>, ZPZV<1>, ZPZV<2), ZPZV<1>, ZPZV<1
03375
                  template<> struct ConwayPolynomial<2, 8> { using ZPZ = aerobus::zpz<2>; using type =
                                                                                                                                                                                       // NOLINT
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1»; };
03377
                 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
           NOLTNT
```

```
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<1»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<2, 11> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
ZPZV<1»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<2, 12> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , Z
                                               ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
03381
                                                                           template<> struct ConwayPolynomial<2, 13> { using ZPZ = aerobus::zpz<2>; using type =
                                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
                                             ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
03382
                                             ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT
                                            template<> struct ConwayPolynomial<2, 15> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
03383
                                            ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>; ZPZV<2; zPzv<1>; ZPZV<2; zpzv<
                                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                             ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
03385
                                                                            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<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1</pre>
03386
                                                                           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<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>,
                                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1»; };</pre>
                                            template<> struct ConwayPolynomial<2, 19> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, 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>, Z
 03388
                                               ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; };
                                             // NOLINT
03389
                                                                          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 =
                                            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 =
 03392
                                            POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; };
                                                                                                                                                                                                                                                                                                                                                                                                                                                                        // NOLINT
                                                                        template<> struct ConwayPolynomial<3, 5> { using ZPZ = aerobus::zpz<3>; using type =
 03393
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT
 03394
                                                                           template<> struct ConwayPolynomial<3, 6> { using ZPZ = aerobus::zpz<3>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<3, 7> { using ZPZ = aerobus::zpz<3>; using type =
 03395
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1»; }; // NOLINT
                                                                          template<> struct ConwayPolynomial<3, 8> { using ZPZ = aerobus::zpz<3>; using type =
 03396
                                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2
                                                                          template<> struct ConwayPolynomial<3, 9> { using ZPZ = aerobus::zpz<3>; using type
                                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; //
                                             NOLINT
                                                                        template<> struct ConwayPolynomial<3, 10> { using ZPZ = aerobus::zpz<3>; using type =
03398
                                             POLYV<2PZV<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 =
                                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>; using type =
03400
                                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1
                                             ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT</pre>
                                                                            template<> struct ConwayPolynomial<3, 13> { using ZPZ = aerobus::zpz<3>; using type
                                             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<2>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<
03402
                                                                              template<> struct ConwayPolynomial<3, 15> { using ZPZ = aerobus::zpz<3>; using type
                                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                             template<> struct ConwayPolynomial<3, 16> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
 03404
                                            ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT
template<> struct ConwayPolynomial<3, 17> { 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»; }; // NOLINT
template<> struct ConwayPolynomial<3, 18> { using ZPZ = aerobus::zpz<3>; using type
                                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              \texttt{ZPZV<2>, \ ZPZV<0>, \ ZPZV<2>, \ ZPZV<1>, \ ZPZV<2>, \ ZPZV<0>, \ ZPZV<0
                                                                            template<> struct ConwayPolynomial<3, 19> { using ZPZ = aerobus::zpz<3>; using type
                                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                               ZPZV<0>, 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<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>,
```

```
ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
03409
                                        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 =
03410
                         POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<5, 3> { using ZPZ = aerobus::zpz<5>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3»; }; // NOLINT
                                        template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
03412
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<5, 5> { using ZPZ = aerobus::zpz<5>; using type =
03413
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<3»; }; // NOLINT
                                          template<> struct ConwayPolynomial<5, 6> { using ZPZ = aerobus::zpz<5>; using type =
03414
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT
03415
                                        template<> struct ConwayPolynomial<5, 7> { using ZPZ = aerobus::zpz<5>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3»; }; // NOLINT
03416
                                        template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<2»; };
                                          template<> struct ConwayPolynomial<5, 9> { using ZPZ = aerobus::zpz<5>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<3»; }; //
03418
                                          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<4>,
                         ZPZV<2»; }; // NOLINT
03419
                                          template<> struct ConwayPolynomial<5, 11> { using ZPZ = aerobus::zpz<5>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                         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<0>, ZPZV<4>, ZPZV<4>,
ZPZV<3>, ZPZV<2>, ZPZV<2»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<5, 13> { using ZPZ = aerobus::zpz<5>; using type =
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<5, 14> { using ZPZ = aerobus::zpz<5>; using type =
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5, ZPZV<5,
03423
                         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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
03425
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<3»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<5, 18> { using ZPZ = aerobus::zpz<5>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                         03427
                         template<> struct ConwayPolynomial<5, 19> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<3»; }; //</pre>
03428
                                        template<> struct ConwayPolynomial<5, 20> { using ZPZ = aerobus::zpz<5>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5, ZP
                          // NOLINT
                                           template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
                         POLYV<ZPZV<1>, ZPZV<4»; }; // NOLINT
                                           template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
                         POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3»; }; // NOLINT
                                        template<> struct ConwayPolynomial<7, 3> { using ZPZ = aerobus::zpz<7>; using type =
03431
                        POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4»; ); // NOLINT template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type =
03432
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; }; // NOLINT
03433
                                        template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT
03434
                                          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 =
03435
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<4»; }; // NOLINT
                                          template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
03436
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<6>, ZPZV<2>, ZPZV<3»; };
03437
                                        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
                                          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 =
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, 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<5>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<0>,
                         ZPZV<5>, ZPZV<0>, ZPZV<3»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<7, 13> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<
```

```
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<5>, ZPZV<5
                             ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<3»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<7, 15> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                            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<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>,
                            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<1>, ZPZV<4»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<7, 18> { using ZPZ = aerobus::zpz<7>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2, ZPZV<6, 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<
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<0>, ZPZV<4»; }; //</pre>
                                                template<> struct ConwayPolynomial<7, 20> { using ZPZ = aerobus::zpz<7>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<6>,
                              ZPZV<2>, ZPZV<5>, ZPZV<2>, ZPZV<3>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3»; };</pre>
                              // NOLINT
                                              template<> struct ConwayPolynomial<11, 1> { using ZPZ = aerobus::zpz<11>; using type =
                            POLYV<ZPZV<1>, ZPZV<9»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<11, 2> { using ZPZ = aerobus::zpz<11>; using type =
                            POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2»; }; // NOLINT
                                                template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<11, 5> { using ZPZ = aerobus::zpz<11>; using type =
03452
 03453
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<0>, ZPZV<9»; }; // NOLINT
 03454
                                             template<> struct ConwayPolynomial<11, 6> { using ZPZ = aerobus::zpz<11>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<6>, 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<0>, ZPZV<4>, ZPZV<9»; }; // NOLINT
 03455
                                                template<> struct ConwayPolynomial<11, 8> { using ZPZ = aerobus::zpz<11>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<8, }}; // NOLINT
                                              template<> struct ConwayPolynomial<11, 9> { using ZPZ = aerobus::zpz<11>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<8
                            NOLINT
                            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>,
03458
                             ZPZV<2»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<10>, ZPZV<9»; }; // NOLINT
    template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type :
03460
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<2>, ZPZV<5>, ZPZV<5>,
                             ZPZV<6>, ZPZV<5>, ZPZV<2»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<11, 13> { using ZPZ = aerobus::zpz<11>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<9»; }; // NOLINT
   template<> struct ConwayPolynomial<11, 14> { using ZPZ = aerobus::zpz<11>; using type =
03462
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                                              template<> struct ConwayPolynomial<11, 15> { using ZPZ = aerobus::zpz<11>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<7>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<10>, ZPZV<2>, ZPZV<2>; }; // NOLINT
03464
                                                 template<> struct ConwayPolynomial<11, 17> { using ZPZ = aerobus::zpz<11>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             template<> struct ConwayPolynomial<11, 18> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<8>, ZPZV<8>, ZPZV<10>, ZPZV<8>,
ZPZV<3>, ZPZV<9>, ZPZV<2>, ZPZV<2>; }; // NOLINT
                                                  template<> struct ConwayPolynomial<11, 19> { 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<0>, ZPZV<0>, ZPZV<8>, ZPZV<2>, ZPZV<9»; };</pre>
                             NOLINT
                            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<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<4>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<2>; };
03468
                                                 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 =
03470
                            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
 03472
                                              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<3>, ZPZV<11»; }; // NOLINT
                           template<> struct ConwayPolynomial<13, 8> { using ZPZ = aerobus::zpz<13>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<12>, ZPZV<2>, ZPZV<3>, ZPZV<2»; }; // NOLINT</pre>
03476
                                                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<
                              // 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<8>, ZPZV<1>, ZPZV<1>,
03478
                             ZPZV<2»: }: // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<13, 11> { using ZPZ = aerobus::zpz<13>; using type =
03479
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<3>, ZPZV<11»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<13, 12> { using ZPZ = aerobus::zpz<13>; using type =
03480
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<4>, ZPZV<4 , ZPZV<4
                                                 template<> struct ConwayPolynomial<13, 13> { using ZPZ = aerobus::zpz<13>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                             ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<11»; }; // NOLINT</pre>
03482
                                                 template<> struct ConwayPolynomial<13, 14> { using ZPZ = aerobus::zpz<13>; using type =
                            POLYY<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<1>; // NOLINT template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type =
03483
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             template<> struct ConwayPolynomial<13, 16> { using ZPZ = aerobus::zpz<13>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<12>,
                            ZPZV<8>, ZPZV<2>, ZPZV<12>, ZPZV<9>, ZPZV<12>, ZPZV<6>, ZPZV<6>, ZPZV<6>; // NOLINT template<> struct ConwayPolynomial<13, 17> { using ZPZ = aerobus::zpz<13>; using type =
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<6>, ZPZV<61>, ZPZV<11»; }; // NOLINT</pre>
                                             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<10>, ZPZV<10>, ZPZV<11>, ZPZV<11>, ZPZV<11>, ZPZV<9>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<0>, ZPZV<9>, ZPZV<2»; }; // NOLINT
                                                template<> struct ConwayPolynomial<13, 19> { using ZPZ = aerobus::zpz<13>; using type =
03487
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0</pre>
                                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5, ZPZV<5,
                              // NOLINT
                                                 template<> struct ConwayPolynomial<17, 1> { using ZPZ = aerobus::zpz<17>; using type =
                            POLYV<ZPZV<1>, ZPZV<14»; }; // NOLINT
03490
                                             template<> struct ConwayPolynomial<17, 2> { using ZPZ = aerobus::zpz<17>; using type =
                           POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
03491
                            POLYVCZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14*; }; // NOLINT template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
03492
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<10>, ZPZV<3»; }; // NOLINT
03493
                                               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 =
03494
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<10>, ZPZV<3>, ZPZV<3>, ZPZV<3>; }; // NOLINT template<> struct ConwayPolynomial<17, 7> { using ZPZ = aerobus::zpz<17>; using type
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<14»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<17, 8> { using ZPZ = aerobus::zpz<17>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<12>, ZPZV<0>, ZPZV<6>, ZPZV<3»; };
03497
                                              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<0>, ZPZV<7>, ZPZV<8>, ZPZV<14*; };
                             // NOLINT
                                                 template<> struct ConwayPolynomial<17, 10> { using ZPZ = aerobus::zpz<17>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>,
                             ZPZV<3»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<17, 11> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14»; }; // NOLINT</pre>
03499
                                                 template<> struct ConwayPolynomial<17, 12> { using ZPZ = aerobus::zpz<17>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<16>, ZPZV<6>, ZPZV<14>, ZPZV<9>, ZPZV<3»; }; // NOLINT
                            template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03501
                            ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<14»; }; // NOLINT
  template<> struct ConwayPolynomial<17, 14> { using ZPZ = aerobus::zpz<17>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
                             ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::zpz<17>; using type =
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<16>, ZPZV<6>, ZPZV<6>, ZPZV<14>, ZPZV<16>, ZPZV<16 , ZPZV<17 , ZPZV<
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<13>,
                              \texttt{ZPZV<5>, ZPZV<2>, ZPZV<12>, ZPZV<13>, ZPZV<12>, ZPZV<1>, ZPZV<3>; }; // \texttt{NOLINT} 
                            template<> struct ConwayPolynomial<17, 17> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<14*; }; // NOLINT
```

```
template<> struct ConwayPolynomial<17, 18> { using ZPZ = aerobus::zpz<17>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<16>
                          ZPZV<7>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<11>, ZPZV<13>, ZPZV<13>, ZPZV<9>, ZPZV<3»; }; // NOLINT</pre>
                                        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<0>, ZPZV<0>, ZPZV<11>, ZPZV<14»; };</pre>
                          NOLINT
                                           template<> struct ConwayPolynomial<17, 20> { using ZPZ = aerobus::zpz<17>; using type
03508
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>,
                          ZPZV<16>, ZPZV<14>, ZPZV<13>, ZPZV<3>, ZPZV<14>, ZPZV<9>, ZPZV<1>, ZPZV<13>, ZPZV<2>, ZPZV<5>,
                          ZPZV<3»; }; // NOLINT
03509
                                           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 =
                          POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2»; }; // NOLINT
03511
                                           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 =
03512
                         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<5>, ZPZV<17»; }; // NOLINT
03514
                                           template<> struct ConwayPolynomial<19, 6> { using ZPZ = aerobus::zpz<19>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<17>, ZPZV<6>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 7> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6 
03515
                                           template<> struct ConwayPolynomial<19, 8> { using ZPZ = aerobus::zpz<19>, using type =
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<10>, ZPZV<3>, ZPZV<2»; };
03517
                                         template<> struct ConwayPolynomial<19, 9> { using ZPZ = aerobus::zpz<19>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<16>, ZPZV<17»; };
                          // NOLINT
                                         template<> struct ConwayPolynomial<19, 10> { using ZPZ = aerobus::zpz<19>; using type =
03518
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<18>, ZPZV<13>, ZPZV<17>, ZPZV<3>, ZPZV<4>,
                          ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<19, 11> { using ZPZ = aerobus::zpz<19>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<8>, ZPZV<17»; }; // NOLINT
   template<> struct ConwayPolynomial<19, 12> { using ZPZ = aerobus::zpz<19>; using type =
03520
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<18>, ZPZV×2>, ZPZV<9>,
                          ZPZV<16>, ZPZV<7>, ZPZV<2»; };</pre>
                                                                                                                                                                 // NOLINT
                                           template<> struct ConwayPolynomial<19, 13> { using ZPZ = aerobus::zpz<19>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          template<> struct ConwayPolynomial<19, 14> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<11>
03522
                          ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<7>, ZPZV<2»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<19, 15> { 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<1>, ZPZV<10>, ZPZ
                         ZPZV<11>, ZPZV<13>, ZPZV<15>, ZPZV<14>, ZPZV<0>, ZPZV<17»; }; // NOLINT
    template<> struct ConwayPolynomial<19, 16> { using ZPZ = aerobus::zpz<19>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0</pre>
03524
                         ZPZV<13>, ZPZV<0>, ZPZV<15>, ZPZV<9>, ZPZV<6>, ZPZV<14>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<19, 17> { using ZPZ = aerobus::zpz<19>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<17»; }; // NOLINT
template<> struct ConwayPolynomial<19, 18> { using ZPZ = aerobus::zpz<19>; using type =
03526
                         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<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<18>, ZPZV<18>, ZPZV<17»; }; //</pre>
                          NOLINT
                         template<> struct ConwayPolynomial<19, 20> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<13>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5, ZPZV<6>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<11>, ZPZV<2»;
03528
                          }; // NOLINT
03529
                                           template<> struct ConwayPolynomial<23, 1> { using ZPZ = aerobus::zpz<23>; using type =
                         POLYV<ZPZV<1>, ZPZV<18»; }; // NOLINT
                                          template<> struct ConwayPolynomial<23, 2> { using ZPZ = aerobus::zpz<23>; using type =
03530
                         POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; }; // NOLINT
                                           template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
03531
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; }; // NOLINT template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
03532
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5s; }; // NOLINT template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type =
03533
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<18»; }; // NOLINT 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
03535
                                           template<> struct ConwayPolynomial<23, 7> { using ZPZ = aerobus::zpz<23>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<18»; }; // NOLINT
03536
                                          template<> struct ConwayPolynomial<23, 8> { using ZPZ = aerobus::zpz<23>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<20>, 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<3>, ZPZV<8>, ZPZV<9>, ZPZV<18»; };
                          // NOLINT
03538
                                          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<5>, ZPZV<6>, ZPZV<6 , ZPZV<6
                          ZPZV<5»: }: // NOLINT
```

```
template<> struct ConwayPolynomial<23, 11> { using ZPZ = aerobus::zpz<23>; using type
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           ZPZV<7>, ZPZV<18»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<23, 12> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<15>, ZPZV<14>, ZPZV<15>, ZPZV<16, ZPZV<16, ZPZV<16, ZPZV<16, ZPZV<16, ZPZV<16, ZPZV<17, ZPZV<18, ZPZV<18, ZPZV<19, 
                                                                                                                                                                                                                                                                              // NOLINT
                                           ZPZV<18>, ZPZV<12>, ZPZV<5»; };</pre>
                                                                        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»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<23, 14> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1 ,
 03542
 03543
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<8>, ZPZV<15>, ZPZV<9>, ZPZV<7>, ZPZV<18>, ZPZV<18*; }; // NOLINT
   template<> struct ConwayPolynomial<23, 16> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
 03544
                                          ZPZV<19>, ZPZV<16>, ZPZV<13>, ZPZV<14>, ZPZV<17>, ZPZV<3, IIIV\(0), IIV\(0), IIIV\(0), IIV\(0), IIV\(
                                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 20>, \ \texttt{ZPZV} < 18»; \ \texttt{}; \ \texttt{} // \ \texttt{NOLINT} 
 03546
                                                                       template<> struct ConwayPolynomial<23, 18> { 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<1>, ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<28, ZPZV<28, ZPZV<18, ZPZV<29, ZPZV<19, ZPZV<29, ZPZV<19, ZPZV<29, ZPZV<19, ZPZV<29, ZPZV
 03547
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            ZPZV<0>, ZPZV<5>, ZPZV<18»; };</pre>
                                           NOLINT
 03548
                                                                       template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type =
                                          POLYV<ZPZV<1>, ZPZV<27»; }; // NOLINT
                                                                     template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
 03549
                                          POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; }; // NOLINT
                                                                       template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<27»; }; // NOLINT
                                                                       template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
  03551
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<2»; }; // NOLINT
                                                                     template<> struct ConwayPolynomial<29, 5> { using ZPZ = aerobus::zpz<29>; using type =
 03552
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<27»; }; // NOLINT
  03553
                                                                       template<> struct ConwayPolynomial<29, 6> { using ZPZ = aerobus::zpz<29>; using type =
                                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2), ZPZV<1>, ZPZV<2, ZPZV<13, ZPZV<13, ZPZV<2, }; // NOLINT template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type =
 03554
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<27»; }; // NOLINT template<> struct ConwayPolynomial<29, 8> { using ZPZ = aerobus::zpz<29>; using type =
  03555
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<23>, ZPZV<28; }; //
  03556
                                                                     template<> struct ConwayPolynomial<29, 9> { using ZPZ = aerobus::zpz<29>; using type
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<22>, ZPZV<22>, ZPZV<27»; };
                                           // NOLINT
 03557
                                                                     template<> struct ConwayPolynomial<29, 10> { using ZPZ = aerobus::zpz<29>; using type =
                                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2
                                           ZPZV<2»; }; // NOLINT
                                                                     template<> struct ConwayPolynomial<29, 11> { using ZPZ = aerobus::zpz<29>; using type =
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<8>, ZPZV<7»; ); // 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<16>, ZPZV<25>,
ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2»; }; // NOLINT
 03559
                                                                   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
                                          ZPZV<U>, ZPZV<U>, ZPZV, LPZV// NODINI
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<3>, ZPZV<1>, ZPZV<3>, ZPZV<10>,
03561
                                           ZPZV<21>, ZPZV<18>, ZPZV<27>, ZPZV<5>, ZPZV<2»; }; // NOLINT</pre>
                                                                        template<> struct ConwayPolynomial<29, 15> { 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<0>, ZPZV<13>,
                                          ZPZV<14>, ZPZV<8>, ZPZV<1>, ZPZV<12>, ZPZV<26>, ZPZV<27»; }; // NOLINT
    template<> struct ConwayPolynomial<29, 16> { using ZPZ = aerobus::zpz<29>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , 
 03563
                                           ZPZV<2>, ZPZV<18>, ZPZV<23>, ZPZV<1>, ZPZV<27>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
                                                                        template<> struct ConwayPolynomial<29, 17> { 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<2>, ZPZV<2>, ZPZV<2 >, // NOLINT
                                          template<> struct ConwayPolynomial<29, 18> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<1>, ZPZV<1 , ZPZV<
 03565
                                          ZPZV<6>, ZPZV<26>, ZPZV<2>, ZPZV<10>, ZPZV<8>, ZPZV<16>, ZPZV<19>, ZPZV<14>, ZPZV<2*; }; // NOLINT
template<> struct ConwayPolynomial<29, 19> { using ZPZ = aerobus::zpz<29>; using type =
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<27»; }; //</pre>
                                           NOLINT
 03567
                                                                    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
  03569
                                                                   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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3»; }; // NOLINT
  03570
```

```
template<> struct ConwayPolynomial<31, 5> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; }; // NOLINT
                                            template<> struct ConwayPolynomial<31, 6> { using ZPZ = aerobus::zpz<31>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<2, ZPZV<1>, ZPZV<3», ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3»; }; // NOLI template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type =
03573
                           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<20>, ZPZV<20
03575
                            // NOLINT
                                               template<> struct ConwayPolynomial<31, 10> { using ZPZ = aerobus::zpz<31>; using type =
03576
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<26>, ZPZV<13>, ZPZV<13>, ZPZV<13>,
                            ZPZV<3»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<31, 11> { using ZPZ = aerobus::zpz<31>; using type =
03577
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<20>, ZPZV<28»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<31, 12> { using ZPZ = aerobus::zpz<31>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<28>, ZPZV<2>, ZPZV<9>,
                            ZPZV<25>, ZPZV<12>, ZPZV<3»; }; // NOLINT</pre>
03579
                                                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»; };</pre>
                                                                                                                                                                                                                            // NOLINT
03580
                                               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<1>, ZPZV<18>, ZPZV<18>, ZPZV<6>, ZPZV<3»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<29>, ZPZV<12>, ZPZV<13>, ZPZV<23>, ZPZV<25>, ZPZV<28»; }; // NOLINT
    template<> struct ConwayPolynomial<31, 16> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
                             ZPZV<24>, ZPZV<26>, ZPZV<28>, ZPZV<11>, ZPZV<19>, ZPZV<27>, ZPZV<3»; }; // NOLINT</pre>
                                           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<0>,
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<28»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<31, 18> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3 , ZPZV<3 ,
03584
                            ZPZV<2>, ZPZV<7>, ZPZV<11>, ZPZV<11>, ZPZV<25>, ZPZV<10>, ZPZV<6>, ZPZV<6>, ZPZV<3»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>
                            NOLINT
                                              template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
03587
                                               template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
                          POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2»; }; // NOLINT
03588
                                             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 =
03589
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2»; }; // NOLINT
                                               template<> struct ConwayPolynomial<37, 5> { using ZPZ = aerobus::zpz<37>; using type =
                            \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<10>, \verb"ZPZV<35"; \verb"}; \verb"// NOLINT" | 
03591
                                             template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<4>, ZPZV<30>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<37, 7> { using ZPZ = aerobus::zpz<37>; using type
03592
                           POLYV<2PZV<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
03593
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<20>, ZPZV<27>, ZPZV<27>, ZPZV<2, ZPZV<2»; }; // NOLINT
03594
                                               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
03595
                                               template<> struct ConwayPolynomial<37, 10> { using ZPZ = aerobus::zpz<37>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<29>, ZPZV<18>, ZPZV<18>, ZPZV<11>, ZPZV<4>,
                            ZPZV<2»; }; // NOLINT</pre>
03596
                                           template<> struct ConwayPolynomial<37, 11> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<2>, ZPZV<35»; }; // NOLINT
                                             template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<10>, ZPZV<23>, ZPZV<23>,
                             ZPZV<18>, ZPZV<33>, ZPZV<2»; }; // NOLINT</pre>
03598
                                           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

template
struct ConwayPolynomial
struct ConwayPolynomial

template
struct ConwayPolynomial
struct ConwayPolynomial

template
struct ConwayPolynomial

template
struct ConwayPolynomial

template
struct ConwayPolynomial
stru
03599
                            ZPZV<32>, ZPZV<16>, ZPZV<1>, ZPZV<9>, ZPZV<2»; }; // NOLINT</pre>
03600
                                             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<1>, ZPZV<31>,
                           ZPZV<28>, ZPZV<27>, ZPZV<13>, ZPZV<34>, ZPZV<33>, ZPZV<35>; }; // NOLINT
    template<> struct ConwayPolynomial<37, 17> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03601
                            template<> struct ConwayPolynomial<37, 18> { using ZPZ = aerobus::zpz<37>; using type
03602
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<8>, ZPZV<8>, ZPZV<15>,
                           ZPZV<1>, ZPZV<22>, ZPZV<20>, ZPZV<12>, ZPZV<32>, ZPZV<14>, ZPZV<27>, ZPZV<20>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<37, 19> { using ZPZ = aerobus::zpz<37>; using type =
03603
```

```
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<36>, ZPZV<36>, ZPZV<23>, ZPZV<35»; }; //</pre>
                        NOLINT
03604
                                      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 =
03605
                       POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6»; }; // NOLINT
                                        template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
 03606
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<35»; }; // NOLINT template<> struct ConwayPolynomial<41, 4> { using ZPZ = aerobus::zpz<41>; using type =
 03607
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<6»; }; // NOLINT
                                      template<> struct ConwayPolynomial<41, 5> { using ZPZ = aerobus::zpz<41>; using type =
 03608
                       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<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<33>, ZPZV<39>, ZPZV<6>, ZPZV<6»; }; // NOLINI
                      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<6>, ZPZV<35»; }; // NOLINT</pre>
 03610
                                       template<> struct ConwayPolynomial<41, 8> { using ZPZ = aerobus::zpz<41>; using type =
03611
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<32>, ZPZV<20>, ZPZV<6>, ZPZV<6>; };
 03612
                                       template<> struct ConwayPolynomial<41, 9> { using ZPZ = aerobus::zpz<41>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<5>, ZPZV<35»; };
                         // NOLINT
03613
                                      template<> struct ConwayPolynomial<41, 10> { using ZPZ = aerobus::zpz<41>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, 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
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<20>, ZPZV<35»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<41, 12> { using ZPZ = aerobus::zpz<41>; using type =
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>
03615
03616
                                        template<> struct ConwayPolynomial<41, 13> { using ZPZ = aerobus::zpz<41>; using type
                       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<12>, ZPZV<12>, ZPZV<15>, ZPZV<4>,
 03617
                       ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6»; }; // NOLINT
   template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35»; }; // NOLINT
template<> struct ConwayPolynomial<41, 17> { using ZPZ = aerobus::zpz<41>; using type =
03619
                       POLYY<ZPZV<0>, ZPZV<0>, ZPZV<3, ZPZV<0>, ZPZV<0 , ZPZV<0 
                                       template<> struct ConwayPolynomial<41, 18> { using ZPZ = aerobus::zpz<41>; using type
03620
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03621
                                      template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type =
                       POLYVCZPZV<1>, ZPZV<0>, ZPZV<0
03622
                                       template<> struct ConwayPolynomial<43, 1> { using ZPZ = aerobus::zpz<43>; using type =
                        POLYV<ZPZV<1>, ZPZV<40»; }; // NOLINT
 03623
                                       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 =
03624
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40»; };
                                                                                                                                                                                                            // NOLINT
                                        template<> struct ConwayPolynomial<43, 4> { using ZPZ = aerobus::zpz<43>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3»; }; // NOLINT
                                        template<> struct ConwayPolynomial<43, 5> { using ZPZ = aerobus::zpz<43>; using type =
 03626
                       template<> struct ConwayPolynomial<43, 6> { using ZPZ = aerobus::zpz<43>; using type =
03627
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<21>, ZPZV<21>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<43, 7> { using ZPZ = aerobus::zpz<43>; using type
 03628
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<7>, ZPZV<7>, ZPZV<40»; }; // NOLINT
03629
                                     template<> struct ConwayPolynomial<43, 8> { using ZPZ = aerobus::zpz<43>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<39>, ZPZV<20>, ZPZV<24>, ZPZV<3»; };
                        NOLINT
                                      template<> struct ConwayPolynomial<43, 9> { using ZPZ = aerobus::zpz<43>; using type =
03630
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<39>, ZPZV<1>, ZPZV<40»; };
                        // NOLINT
                                       template<> struct ConwayPolynomial<43, 10> { using ZPZ = aerobus::zpz<43>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<26>, ZPZV<36>, ZPZV<5>, ZPZV<27>, ZPZV<24>,
                         ZPZV<3»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type =
03632
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<1>, ZPZV<10>, ZPZ
                                     template<> struct ConwayPolynomial<43, 12> { using ZPZ = aerobus::zpz<43>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<27>, ZPZV<16>, ZPZV<17>, ZPZV<6>, ZPZV<23>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38
03634
                                      template<> struct ConwayPolynomial<43, 13> { using ZPZ = aerobus::zpz<43>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<40»; };</pre>
                                                                                                                                                                                      // NOLINT
                                      template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>,
                        \mbox{ZPZV}<37>, \mbox{ZPZV}<18>, \mbox{ZPZV}<4>, \mbox{ZPZV}<19>, \mbox{ZPZV}<3»; }; // \mbox{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<2>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3</pre>
 03636
```

```
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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
                           \verb|ZPZV<0>, \verb|ZPZV<0>, \verb|ZPZV<0>, \verb|ZPZV<0>, \verb|ZPZV<0>, \verb|ZPZV<0>, \verb|ZPZV<36>, \verb|ZPZV<40>, \verb|ZPZV<40>, || // \verb|NOLINT| | // | NOLINT| | // 
03638
                          template<> struct ConwayPolynomial<43, 18> \{ using ZPZ = aerobus:: zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<28>, ZPZV<41>,
                          ZPZV<24>, ZPZV<7>, ZPZV<24>, ZPZV<29>, ZPZV<16>, ZPZV<34>, ZPZV<37>, ZPZV<18>, ZPZV<3»; }; // NOLINT
                                             template<> struct ConwayPolynomial<43, 19> { using ZPZ = aerobus::zpz<43>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40»; }; //</pre>
                          NOLINT
03640
                                          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 =
                          POLYV<ZPZV<1>, ZPZV<45>, ZPZV<5»; }; // NOLINT
03642
                                            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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; }; // NOLINT
03643
                                           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
03645
                                            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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT
03646
                                             template<> struct ConwayPolynomial<47, 8> { using ZPZ = aerobus::zpz<47>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<19>, ZPZV<3>, ZPZV<5»; };
03648
                                         template<> struct ConwayPolynomial<47, 9> { using ZPZ = aerobus::zpz<47>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<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>,
03649
                           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»; }; // NOLINT
    template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type =
03651
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<46>, ZPZV<46>, ZPZV<40>, ZPZV<35>, ZPZV<35>, ZPZV<35
                          ZPZV<14>, ZPZV<9>, ZPZV<5»; };</pre>
                                                                                                                                                                     // NOLINT
                                          template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          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<36>, ZPZV<36>, ZPZV<30>, ZPZV<30>
03653
                          ZPZV<17>, ZPZV<24>, ZPZV<9>, ZPZV<32>, ZPZV<5>; }; // NOLINT
template<> struct ConwayPolynomial<47, 15> { using ZPZ = aerobus::zpz<47>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<31>, ZPZV<14>, ZPZV<42>, ZPZV<13>, ZPZV<17>, ZPZV<42»; }; // NOLINT
    template<> struct ConwayPolynomial<47, 17> { using ZPZ = aerobus::zpz<47>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03655
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<42»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<47, 18> { using ZPZ = aerobus::zpz<47>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<41>, ZPZV<42>,
                          ZPZV<26>, ZPZV<44>, ZPZV<24>, ZPZV<22>, ZPZV<11>, ZPZV<5>, ZPZV<45>, ZPZV<33>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<47, 19> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<3>, ZPZV<42»; }; //
03657
                                             template<> struct ConwayPolynomial<53, 1> { using ZPZ = aerobus::zpz<53>; using type =
                          POLYV<ZPZV<1>, ZPZV<51»; }; // NOLINT
                                          template<> struct ConwayPolynomial<53, 2> { using ZPZ = aerobus::zpz<53>; using type =
03659
                         POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2»; }; // NOLINT
03660
                                            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 =
03661
                          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<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT</pre>
03662
                                          template<> struct ConwayPolynomial<53, 6> { using ZPZ = aerobus::zpz<53>; using type =
03663
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<4>, ZPZV<45>, ZPZV<2»; };
                                            template<> struct ConwayPolynomial<53, 7> { using ZPZ = aerobus::zpz<53>; using type =
03664
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51»; }; // NOLINT
03665
                                           template<> struct ConwayPolynomial<53, 8> { using ZPZ = aerobus::zpz<53>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<1>, ZPZV<2»; };
03666
                                           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<0>, ZPZV<1>, ZPZV<5>, ZPZV<51»; };
                          // NOLINT
03667
                                           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<27>, ZPZV<15>, ZPZV<29>,
                          ZPZV<2»: 1: // NOLINT
                                             template<> struct ConwayPolynomial<53, 11> { using ZPZ = aerobus::zpz<53>; using type =
03668
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<15>, ZPZV<51»; }; // NOLINT</pre>
03669
                                         template<> struct ConwayPolynomial<53, 12> { using ZPZ = aerobus::zpz<53>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<44>, ZPZV<13>, ZPZV<41>, ZPZV<42>, ZPZV<44>, ZPZV<44>, ZPZV<41>, ZPZV<
03670
                                         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<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<50>, ZPZV<5
                              ZPZV<0>, ZPZV<37>, ZPZV<12>, ZPZV<23>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<53, 15> { using ZPZ = aerobus::zpz<53>; using type =
03672
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<53, 17> { using ZPZ = aerobus::zpz<53>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03674
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>,
                               ZPZV<27>, ZPZV<0>, ZPZV<39>, ZPZV<44>, ZPZV<6>, ZPZV<8>, ZPZV<16>, ZPZV<11>, ZPZV<2»; }; // NOLINT</pre>
                                                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<1>, ZPZV<11>, ZPZV<51»; }; //</pre>
                               NOLINT
                                                   template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
                              POLYV<ZPZV<1>, ZPZV<57»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<59, 2> { using ZPZ = aerobus::zpz<59>; using type =
                              POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2»; }; // NOLINT
03678
                                                 template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; }; // NOLINT template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
03679
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<2»; }; // NOLINT
03680
                                                template<> struct ConwayPolynomial<59, 5> { using ZPZ = aerobus::zpz<59>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<8>, ZPZV<57»; }; // NOLINT
03681
                                                   template<> struct ConwayPolynomial<59, 6> { using ZPZ = aerobus::zpz<59>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18>, ZPZV<38>, ZPZV<0>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<59, 7> { using ZPZ = aerobus::zpz<59>; using type =
03682
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<57»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<59, 8> { using ZPZ = aerobus::zpz<59>; using type
03683
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<32>, ZPZV<2>, ZPZV<50>, ZPZV<50>, ZPZV<2»; }; //
                               NOLINT
03684
                                                 template<> struct ConwayPolynomial<59, 9> { using ZPZ = aerobus::zpz<59>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<47>, ZPZV<57»; };
                                                   template<> struct ConwayPolynomial<59, 10> { using ZPZ = aerobus::zpz<59>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28>, ZPZV<25>, ZPZV<4>, ZPZV<39>, ZPZV<15>,
                               ZPZV<2»; }; // NOLINT</pre>
03686
                                                template<> struct ConwayPolynomial<59, 11> { using ZPZ = aerobus::zpz<59>; using type =
                              POLYYCZPZV<1>, ZPZV<0>, ZPZV<0
                              template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<21>, ZPZV<38>,
                               ZPZV<8>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<59, 13> { using ZPZ = aerobus::zpz<59>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57»; };</pre>
                                                                                                                                                                                                                                                 // NOLINT
                                                   template<> struct ConwayPolynomial<59, 14> { using ZPZ = aerobus::zpz<59>; using type =
03689
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<3>, 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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, 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 =
03690
                               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57»; };</pre>
03692
                                                    template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59>; using type
                              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<32>, ZPZV
03693
                               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<57»; }; //</pre>
                               NOLINT
03694
                                                   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 =
03695
                             POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
03696
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; }; // NOLINT template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
03697
                             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<12>, ZPZV<59»; }; // NOLINT
03698
                                                 template<> struct ConwayPolynomial<61, 6> { using ZPZ = aerobus::zpz<61>; using type =
03699
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<3>, ZPZV<29>, 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<2>, ZPZV<59»; }; // NOLINT
03700
                                                  template<> struct ConwayPolynomial<61, 8> { using ZPZ = aerobus::zpz<61>; using type =
03701
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<1>, ZPZV<56>, ZPZV<2»; };
                              template<> struct ConwayPolynomial<61, 9> { using ZPZ = aerobus::zpzc61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<50>,
                               // NOLINT
                              template<> struct ConwayPolynomial<61, 10> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<15>, ZPZV<44>, ZPZV<16>, ZPZV<6>,
03703
```

```
ZPZV<2»; };</pre>
                                            template<> struct ConwayPolynomial<61, 11> { using ZPZ = aerobus::zpz<61>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<18>, ZPZV<59»; }; // NOLINT</pre>
03705
                                             template<> struct ConwayPolynomial<61, 12> { using ZPZ = aerobus::zpz<61>; using type =
                          POLYY<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<42>, ZPZV<42>, ZPZV<43>, ZPZV<38>, ZPZV<38>, ZPZV<14>, ZPZV<15>, ZPZV<2»; }; // NOLINT
                                              template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<59»; }; // NOLINT</pre>
                          ZPZV<U>, ZPZV<U>, ZPZV<U>, ZPZV<U>, ZPZV<U>, ZPZV<U>; ;; // NOLLN1
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<48>, ZPZV<48>, ZPZV<46>, ZPZV<11>,
03707
                           ZPZV<8>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<61, 15> { using ZPZ = aerobus::zpz<61>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<35>, ZPZV<44>, ZPZV<25>, ZPZV<23>, ZPZV<51>, ZPZV<59»; }; // NOLINT
    template<> struct ConwayPolynomial<61, 17> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                            \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0
                                             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<35>, ZPZV<35>, ZPZV<36>, ZPZV<13>
                            \texttt{ZPZV<36>, \ ZPZV<4>, \ ZPZV<32>, \ ZPZV<57>, \ ZPZV<42>, \ ZPZV<25>, \ ZPZV<25>, \ ZPZV<52>, \ ZPZV<52>, \ ZPZV<50>, \ ZPZV<28+; \ \}; \ // \ \texttt{NOLINT} } 
                          template<> struct ConwayPolynomial<61, 19> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZ
                                            template<> struct ConwayPolynomial<67, 1> { using ZPZ = aerobus::zpz<67>; using type =
                          POLYV<ZPZV<1>, ZPZV<65»; }; // NOLINT
                                             template<> struct ConwayPolynomial<67, 2> { using ZPZ = aerobus::zpz<67>; using type =
03713
                         POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
03714
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65»; }; // NOLINT
                                            template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type =
03715
                          template<> struct ConwayPolynomial<67, 5> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<65»; }; // NOLINT</pre>
03716
                                           template<> struct ConwayPolynomial<67, 6> { using ZPZ = aerobus::zpz<67>; using type =
03717
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<63>, ZPZV<49>, ZPZV<55>, ZPZV<2»; }; // NOLINT
03718
                                             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<7>, ZPZV<65»; }; // NOLINT
03719
                                          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<17>, ZPZV<64>, ZPZV<64>; };
                           NOLINT
03720
                                           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<25>, ZPZV<49>, ZPZV<55>, ZPZV<65»; };
                            // NOLINT
03721
                                           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
                                             template<> struct ConwayPolynomial<67, 11> { using ZPZ = aerobus::zpz<67>; using type =
03722
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<9>, ZPZV<65»; }; // NOLINT</pre>
03723
                                           template<> struct ConwayPolynomial<67, 12> { using ZPZ = aerobus::zpz<67>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57>, ZPZV<27>, ZPZV<4>, ZPZV<64>, ZPZV<64>, ZPZV<64>, ZPZV<21>, ZPZV<27>, ZPZV<27>, ZPZV<28; }; // NOLINT
03724
                                             template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<22>, ZPZV<5>,
                          ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<2>; }; // NOLINT
template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>,
                          ZPZV<52>, ZPZV<41>, ZPZV<20>, ZPZV<21>, ZPZV<46>, ZPZV<65»; }; // NOLINT
    template<> struct ConwayPolynomial<67, 17> { using ZPZ = aerobus::zpz<67>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<65»; }; // NOLINT
template<> struct ConwayPolynomial<67, 18> { using ZPZ = aerobus::zpz<67>; using type =
03728
                          POLYYCZPZVC1>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC5>, ZPZVC5
                                           template<> struct ConwayPolynomial<67, 19> { using ZPZ = aerobus::zpz<67>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<65»; }; //</pre>
                           NOLINT
                                            template<> struct ConwayPolynomial<71, 1> { using ZPZ = aerobus::zpz<71>; using type =
                          POLYV<ZPZV<1>, ZPZV<64»; }; // NOLINT
                                           template<> struct ConwayPolynomial<71, 2> { using ZPZ = aerobus::zpz<71>; using type =
03731
                          POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7»; }; // NOLINT
                                           template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
03732
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; ); // NOLINT template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
03733
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, ZPZV<7»; };
                                                                                                                                                                                                                                                                                       // NOLINT
                                             template<> struct ConwayPolynomial<71, 5> { using ZPZ = aerobus::zpz<71>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<64»; }; // NOLINT
03735
                                          template<> struct ConwayPolynomial<71, 6> { using ZPZ = aerobus::zpz<71>; using type =
                        POLYV<2PZV<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 =
03736
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<64»; };
                                                            template<> struct ConwayPolynomial<71, 8> { using ZPZ = aerobus::zpz<71>; using type
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<22>, ZPZV<19>, ZPZV<7»; }; //
                                   NOLINT
                                   template<> struct ConwayPolynomial<71, 9> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43>, ZPZV<43>, ZPZV<62>, ZPZV<64»; };</pre>
 03738
                                                            template<> struct ConwayPolynomial<71, 10> { using ZPZ = aerobus::zpz<71>; using type
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<53>, ZPZV<17>, ZPZV<26>, ZPZV<14>, 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>,
 03740
                                                                                                                                                                                 // NOLINT
                                    ZPZV<48>, ZPZV<64»; };
                                                            template<> struct ConwayPolynomial<71, 12> { using ZPZ = aerobus::zpz<71>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<28>, ZPZV<29>, ZPZV<25, ZPZV<21>,
                                     ZPZV<58>, ZPZV<23>, ZPZV<7»; }; // NOLINT</pre>
                                                            template<> struct ConwayPolynomial<71, 13> { using ZPZ = aerobus::zpz<71>; using type =
                                   POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     \mbox{ZPZV<32>, ZPZV<18>, ZPZV<52>, ZPZV<67>, ZPZV<49>, ZPZV<64*); }; // \mbox{NOLINT} 
                                   template<> struct ConwayPolynomial<71, 17> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
                                   ZPZV<0>, ZPZV<0>
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   ZPZV<0>,
                                     ZPZV<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</pre>, ZPZV<0>, ZP
                                    NOLINT
 03746
                                                          template<> struct ConwayPolynomial<73, 1> { using ZPZ = aerobus::zpz<73>; using type =
                                  POLYV<ZPZV<1>, ZPZV<68»; }; // NOLINT
                                                             template<> struct ConwayPolynomial<73, 2> { using ZPZ = aerobus::zpz<73>; using type =
                                   POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5»; }; // NOLINT
  03748
                                                       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 =
  03749
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<58>, ZPZV<58; }; // NOLINT template<> struct ConwayPolynomial<73, 5> { using ZPZ = aerobus::zpz<73>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<68»; }; // NOLINT
                                                          template<> struct ConwayPolynomial<73, 6> { using ZPZ = aerobus::zpz<73>; using type =
  03751
                                  POLYV<2PZV<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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; 
 03752
 03753
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<53>, ZPZV<39>, ZPZV<18>, ZPZV<5»; };
                                    NOLINT
 03754
                                                       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<15>, ZPZV<15>, ZPZV<68»; };
                                     // NOLINT
                                                          template<> struct ConwayPolynomial<73, 10> { using ZPZ = aerobus::zpz<73>; using type =
03755
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<23>, ZPZV<33>, ZPZV<32>, ZPZV<69>,
                                    ZPZV<5»; }; // NOLINT</pre>
                                                       template<> struct ConwayPolynomial<73, 11> { using ZPZ = aerobus::zpz<73>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<68»; }; // NOLINT
 03757
                                                            template<> struct ConwayPolynomial<73, 12> { using ZPZ = aerobus::zpz<73>; using type =
                                   POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<69>, ZPZV<69>, ZPZV<52>, ZPZV<26>, ZPZV<26>, ZPZV<46>, ZPZV<46>, ZPZV<46>, ZPZV<52>, ZPZV<52>, ZPZV<55, ZPZV<
                                                          template<> struct ConwayPolynomial<73, 13> { using ZPZ = aerobus::zpz<73>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   template<> struct ConwayPolynomial<73, 15> { using ZPZ = aerobus::zpz<73>; using type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>,
                                   ZPZV<10>, ZPZV<33>, ZPZV<57>, ZPZV<57>, ZPZV<62>, ZPZV<68»; }; // NOLINT
   template<> struct ConwayPolynomial<73, 17> { using ZPZ = aerobus::zpz<73>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    template<> struct ConwayPolynomial<73, 19> { using ZPZ = aerobus::zpz<73>; using type =
                                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     ZPZV<0>, ZPZV<0>
                                                        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 =
 03763
                                   POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3»; }; // NOLINT
                                                             template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76»; }; // NOLINT
template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
  03765
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<79, 5> { using ZPZ = aerobus::zpz<79>; using type =
 03766
                                   POLYV<ZPZV<1>, 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 template<> struct ConwayPolynomial<79, 7> { using ZPZ = aerobus::zpz<79>; using type =
  03768
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<76»; }; // NOLINT
                                   template<> struct ConwayPolynomial<79, 8> { using ZPZ = aerobus::zpz<79>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<60>, ZPZV<59>, ZPZV<48>, ZPZV<48>, ZPZV<3»; }; //</pre>
  03769
```

```
NOLINT
                          template<> struct ConwayPolynomial<79, 9> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<57>, ZPZV<19>, ZPZV<76»; };
03770
                           // NOLINT
                          template<> struct ConwayPolynomial<79, 10> { using ZPZ = aerobus::zpz<79>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<51>, ZPZV<1>, ZPZV<1>, ZPZV<30>, ZPZV<42>,
                          ZPZV<3»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<79, 11> { using ZPZ = aerobus::zpz<79>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<3>, ZPZV<76»; }; // NOLINT
   template<> struct ConwayPolynomial<79, 12> { using ZPZ = aerobus::zpz<79>; using type =
03773
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<45>, ZPZV<45>, ZPZV<52>, ZPZV<40>,
                          ZPZV<59>, ZPZV<62>, ZPZV<3»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<78>, ZPZV<4>, ZPZV<76»; }; // NOLINT
   template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                            \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 25>, \ \texttt{ZPZV} < 76»; \ \}; \ // \ \texttt{NOLINT} 
                                           template<> struct ConwayPolynomial<79, 19> { using ZPZ = aerobus::zpz<79>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            \texttt{ZPZV} < \texttt{0} >, \ \texttt{ZPZV} < \texttt{25} >, \ \texttt{ZPZV} < \texttt{76} *; \ \}; \quad // 
                           NOLINT
                                            template<> struct ConwayPolynomial<83, 1> { using ZPZ = aerobus::zpz<83>; using type =
                         POLYV<ZPZV<1>, ZPZV<81»; }; // NOLINT
                                            template<> struct ConwayPolynomial<83, 2> { using ZPZ = aerobus::zpz<83>; using type =
                         POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; }; // NOLINT
03779
                                           template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT template<> struct ConwayPolynomial<83, 4> { using ZPZ = aerobus::zpz<83>; using type =
03780
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<83, 5> { using ZPZ = aerobus::zpz<83>; using type =
03781
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<81»; }; // NOLINT
03782
                                         template<> struct ConwayPolynomial<83, 6> { using ZPZ = aerobus::zpz<83>; using type =
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<32>, ZPZV<17, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<83, 7> { using ZPZ = aerobus::zpz<83>; using type =
03783
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>; zPZV<3>; zPZV<8>; ); // 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»; };
03785
                                          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<1>, ZPZV<24>, ZPZV<24>, ZPZV<18>, ZPZV<81»; };
                           // NOLINT
03786
                                             template<> struct ConwayPolynomial<83, 10> { using ZPZ = aerobus::zpz<83>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5, ZPZV
                           ZPZV<2»; }; // NOLINT</pre>
03787
                                          template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type =
                           \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                           ZPZV<17>, ZPZV<81»: }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type =
03788
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<12>, ZPZV<31>, ZPZV<19>, ZPZV<65>, ZPZV<55>, ZPZV<75>, ZPZV<75>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            template<> struct ConwayPolynomial<83, 17> { using ZPZ = aerobus::zpz<83>; using type
03790
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<81»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<81»; }; //
                          NOLINT
                                             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 =
                         POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; }; // NOLINT
03794
                                            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 =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<89, 5> { using ZPZ = aerobus::zpz<89>; using type =
03796
                         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 =
03797
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<82>, ZPZV<80>, ZPZV<15>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type
03798
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<86»; }; // NOLINT
                                          template<> struct ConwayPolynomial<89, 8> { using ZPZ = aerobus::zpz<89>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<40>, ZPZV<79>, ZPZV<3»; }; //
                          NOLINT
                         template<> struct ConwayPolynomial<89, 9> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<1>, ZPZV<1>, ZPZV<6>, ZPZV<86»; };
03800
                           // NOLINT
                          template<> struct ConwayPolynomial<89, 10> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<33>, ZPZV<82>, ZPZV<52>, ZPZV<4>,
                          ZPZV<3»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03802
```

```
ZPZV<26>, ZPZV<86»; };</pre>
                              template<> struct ConwayPolynomial<89, 12> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<85>, ZPZV<15>, ZPZV<44>, ZPZV<51>, ZPZV<8>,
                               ZPZV<70>, ZPZV<52>, ZPZV<3»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type
03804
                              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                   template<> struct ConwayPolynomial<89, 17> { using ZPZ = aerobus::zpz<89>; using type
03805
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               template<> struct ConwayPolynomial<89, 19> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
03806
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3+, ZPZV<34+, ZPZV<86*; }; //</pre>
03807
                                                 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 =
03808
                              POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<97, 3> { using ZPZ = aerobus::zpz<97>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<92»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<97, 4> { using ZPZ = aerobus::zpz<97>; using type =
03810
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<97, 5> { using ZPZ = aerobus::zpz<97>; using type =
03811
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<3>, ZPZV<92»; }; // NOLINT template<> struct ConwayPolynomial<97, 6> { using ZPZ = aerobus::zpz<97>; using type =
03812
                              POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<5>, ZPZY<58>, ZPZY<58>, ZPZY<58>, ZPZY<58>, ZPZY<58>; ); // NOLINT template<> struct ConwayPolynomial<97, 7> { using ZPZ = aerobus::zpz<97>; using type =
03813
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<92»; }; // NOLINT
                             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»; };</pre>
03814
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            // NOLINT
                                                 template<> struct ConwayPolynomial<97, 9> { using ZPZ = aerobus::zpz<97>; using type
03815
                              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<59>, ZPZV<7>, ZPZV<92»; };
                                                template<> struct ConwayPolynomial<97, 10> { using ZPZ = aerobus::zpz<97>; using type =
03816
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<20>,
                               ZPZV<5»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type =
03817
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<5>, ZPZV<92»; };</pre>
                                                                                                                                                      // 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<5»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<97, 13> { using ZPZ = aerobus::zpz<97>; using type =
03819
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<92»; };</pre>
                                                                                                                                                                                                                                                // NOLINT
03820
                                                 template<> struct ConwayPolynomial<97, 17> { using ZPZ = aerobus::zpz<97>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               template<> struct ConwayPolynomial<97, 19> { using ZPZ = aerobus::zpz<97>; using type =
03821
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>
03822
                                                  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 =
03823
                              POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<99»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
03825
                              template<> struct ConwayPolynomial<101, 5> { using ZPZ = aerobus::zpz<101>; using type =
03826
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<99»; }; // NOLINT
03827
                                                   template<> struct ConwayPolynomial<101, 6> { using ZPZ = aerobus::zpz<101>; using type =
                              POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<90>, ZPZV<20>, ZPZV<67>, ZPZV<2*; }; // NOLINT
03828
                                                template<> struct ConwayPolynomial<101, 7> { using ZPZ = aerobus::zpz<101>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
03829
                              NOLINT
                                                    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»; };
                                // NOLINT
                              template<> struct ConwayPolynomial<101, 10> { using ZPZ = aerobus::zpz<101>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<67>, ZPZV<49>, ZPZV<100>, ZPZV<100>, ZPZV<52>,
03831
                               ZPZV<2»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<101, 11> { using ZPZ = aerobus::zpz<101>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<31>, ZPZV<99»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<101, 12> { using ZPZ = aerobus::zpz<101>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<64>, ZPZV<64>, ZPZV<39>, ZPZV<78>, ZPZV<48>, ZPZV<84>, ZPZV<21>, ZPZV<22>; }; // NOLINT
                                                    template<> struct ConwayPolynomial<101, 13> { using ZPZ = aerobus::zpz<101>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<99»; };</pre>
                                                                                                                                                                                                                                                 // NOLINT
                              template<> struct ConwayPolynomial<101, 17> { using ZPZ = aerobus::zpz<101>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03835
```

```
template<> struct ConwayPolynomial<101, 19> { using ZPZ = aerobus::zpz<101>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>
                               NOLINT
03837
                                                   template<> struct ConwayPolynomial<103, 1> { using ZPZ = aerobus::zpz<103>; using type =
                             POLYV<ZPZV<1>, ZPZV<98»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<103, 2> { using ZPZ = aerobus::zpz<103>; using type =
                              POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<103, 3> { using ZPZ = aerobus::zpz<103>; using type =
03839
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<98»; }; // NOLINT template<> struct ConwayPolynomial<103, 4> { using ZPZ = aerobus::zpz<103>; using type =
03840
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<88>, ZPZV<5»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
03841
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<98»; }; // NOLINT
03842
                                                 template<> struct ConwayPolynomial<103, 6> { using ZPZ = aerobus::zpz<103>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<96>, ZPZV<30>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<103, 7> { using ZPZ = aerobus::zpz<103>; using type =
03843
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
                                                    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»; }; //
03845
                                                  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<97>, ZPZV<97>, ZPZV<51>, ZPZV<98»; };
                               // NOLINT
03846
                                                    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>,
                               ZPZV<5»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<103, 11> { using ZPZ = aerobus::zpz<103>; using type =
                               \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                               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<94>, ZPZV<20>, ZPZV<81>, ZPZV<29>, ZPZV<88>, ZPZV<88 , ZPZV<
                                                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
                                                   template<> struct ConwayPolynomial<103, 17> { using ZPZ = aerobus::zpz<103>; using type =
03850
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 102>, \ \texttt{ZPZV} < 8>, \ \texttt{ZPZV} < 98»; \ \}; \ // \ \texttt{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<2>, ZPZV<3</pre>; ;;
                               NOLINT
03852
                                                   template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
                              POLYV<ZPZV<1>, ZPZV<105»; }; // NOLINT
03853
                                                    template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
                              POLYV<ZPZV<1>, ZPZV<103>, ZPZV<2»; }; // NOLINT
03854
                                                  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 =
03855
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<107, 5> { using ZPZ = aerobus::zpz<107>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<105»; }; // NOLINT
                            template<> struct ConwayPolynomial<107, 6> { using ZPZ = aerobus::zpz<107>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<52>, ZPZV<22>, ZPZV<79>, ZPZV<2»; }; // NOLINT</pre>
03857
                                                    template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type
03858
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<105»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<107, 8> { using ZPZ = aerobus::zpz<107>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<24>, ZPZV<95>, ZPŽV<2»; };
                               NOLINT
                                                  template<> struct ConwayPolynomial<107, 9> { using ZPZ = aerobus::zpz<107>; using type =
03860
                               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<66>, ZPZV<105»; };
                                // NOLINT
                                                    template<> struct ConwayPolynomial<107, 10> { using ZPZ = aerobus::zpz<107>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<94>, ZPZV<61>, ZPZV<83>, ZPZV<83>, ZPZV<85>,
                               ZPZV<2»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<107, 11> { using ZPZ = aerobus::zpz<107>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
03862
                                                     template<> struct ConwayPolynomial<107, 12> { using ZPZ = aerobus::zpz<107>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<61>, ZPZV<48>, ZPZV<66>, ZPZV<61>, ZPZV<61>, ZPZV<42>, ZPZV<57>, ZPZV<2»; }; // NOLINT
                              template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03864
                               ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105»; };</pre>
                                                                                                                                                                                                                                                         // NOLINT
                                                     template<> struct ConwayPolynomial<107, 17> { using ZPZ = aerobus::zpz<107>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                                                                                                 // NOLINT
                                                    template<> struct ConwayPolynomial<107, 19> { using ZPZ = aerobus::zpz<107>; using type =
                              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                    template<> struct ConwayPolynomial<109, 1> { using ZPZ = aerobus::zpz<109>; using type =
                            POLYV<ZPZV<1>, ZPZV<103»; }; // NOLINT
                                                template<> struct ConwayPolynomial<109, 2> { using ZPZ = aerobus::zpz<109>; using type =
03868
                            POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
03869
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT
                    template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; }; // NOLINT
                                  template<> struct ConwayPolynomial<109, 5> { using ZPZ = aerobus::zpz<109>; using type =
 03871
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103»; }; // NOLINT
03872
                                   template<> struct ConwayPolynomial<109, 6> { using ZPZ = aerobus::zpz<109>; using type =
                    POLYV<2P2V<1>, 2P2V<0>, ZP2V<0>, ZP2V<107>, ZP2V<102>, ZP2V<66>, ZP2V<66>; }; // NOLINT
                                   template<> struct ConwayPolynomial<109, 7> { using ZPZ = aerobus::zpz<109>; using type =
 03873
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<103»; }; // NOLINT template<> struct ConwayPolynomial<109, 8> { using ZPZ = aerobus::zpz<109>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<34>, ZPZV<34>, ZPZV<86>; };
 03874
                     NOLINT
03875
                                  template<> struct ConwayPolynomial<109, 9> { using ZPZ = aerobus::zpz<109>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<93>, ZPZV<93>, ZPZV<103»; };
                      // NOLINT
03876
                                   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
                                    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>
03878
                                   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<53>, ZPZV<37>, ZPZV<65>, ZPZV<103>, ZPZV<28>, ZPZV<69; }; // NOLINT
03879
                                   template<> struct ConwayPolynomial<109, 13> { using ZPZ = aerobus::zpz<109>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<103»; };</pre>
                                                                                                                                                                       // NOLINT
                                 template<> struct ConwayPolynomial<109, 17> { using ZPZ = aerobus::zpz<109>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT
template<> struct ConwayPolynomial<109, 19> { using ZPZ = aerobus::zpz<109>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<15</pre>
03882
                                   template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
                     POLYV<ZPZV<1>, ZPZV<110»; }; // NOLINT
                                  template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
03883
                     POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3»; }; // NOLINT
                                   template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
                     POLYY<ZPZY<1>, ZPZY<0>, ZPZY<8>, ZPZY<110»; }; // NOLINT template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
 03885
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<62>, ZPZV<3>; }; // NOLINT
template<> struct ConwayPolynomial<113, 5> { using ZPZ = aerobus::zpz<113>; using type =
 03886
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<710»; }; // NOLINT template<> struct ConwayPolynomial<113, 6> { using ZPZ = aerobus::zpz<113>; using type =
                                                                                                                                                                                                                                                                                              // NOLINT
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<59>, ZPZV<30>, ZPZV<71>, ZPZV<3»; };
 03888
                                 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<110»; }; // NOLINT
03889
                                 template<> struct ConwayPolynomial<113, 8> { using ZPZ = aerobus::zpz<113>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<98>, ZPZV<38>, ZPZV<28>, ZPZV<3»; }; //
                                   template<> struct ConwayPolynomial<113, 9> { using ZPZ = aerobus::zpz<113>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<87>, ZPZV<87>, ZPZV<71>, ZPZV<110»; };
                      // NOLINT
03891
                                  template<> struct ConwayPolynomial<113, 10> { using ZPZ = aerobus::zpz<113>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>,
                      ZPZV<3»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<3>, ZPZV<110»; }; // NOLINT
template<> struct ConwayPolynomial<113, 12> { using ZPZ = aerobus::zpz<113>; using type =
03893
                     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
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110»; }; // NOLINT</pre>
03895
                                   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<4>, ZPZV<4110»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<113, 19> { 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<0>, ZPZV<22>, ZPZV<110»; }; //</pre>
                     NOLINT
03897
                                  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 =
                     POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3»; }; // NOLINT
 03899
                                 template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124»; };
                                                                                                                                                                                            // NOLINT
                                  template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type =
03900
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<9>, 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
 03902
                                 template<> struct ConwayPolynomial<127, 6> { using ZPZ = aerobus::zpz<127>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<84>, 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<15>, ZPZV<124»; }; // NOLI
 03903
```

```
template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<104>, ZPZV<55>, ZPZV<8>, ZPZV<3»; }; //
                         NOLINT
                        template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<119>, ZPZV<126>, ZPZV<124»;</pre>
 03905
                         }; // NOLINT
                                          template<> struct ConwayPolynomial<127, 10> { using ZPZ = aerobus::zpz<127>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<64>, ZPZV<95>, ZPZV<60, ZPZV<4>,
                          ZPZV<3»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<124»; }; // NOLINT
03907
                                         template<> struct ConwayPolynomial<127, 12> { using ZPZ = aerobus::zpz<127>; using type
03908
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<119>, ZPZV<25>, ZPZV<33>, ZPZV<9¬, ZPZV<15>,
                         ZPZV<99>, ZPZV<8>, ZPZV<3»; }; // NOLINT</pre>
03909
                                       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
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<124»; };</pre>
03911
                                         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>, 
                         NOLINT
                                          template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
                        POLYV<ZPZV<1>, ZPZV<129»; }; // NOLINT
03913
                                        template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
                        POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2»; }; // NOLINT
                                       template<> struct ConwayPolynomial<131, 3> { using ZPZ = aerobus::zpz<131>; using type =
03914
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<129»; }; // NOLINT template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type =
 03915
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2»; }; // NOLINT
 03916
                                      template<> struct ConwayPolynomial<131, 5> { using ZPZ = aerobus::zpz<131>; using type =
                         \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<19>, \verb"ZPZV<129"; \verb"}; $ // \verb"NOLINT" | NOLINT" | NOLINT | NOL
 03917
                                         template<> struct ConwayPolynomial<131, 6> { using ZPZ = aerobus::zpz<131>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<4>, ZPZV<2>, ZPZV<2>, ZPZV<2), ZPZV<2; } ; // NOLINT template<> struct ConwayPolynomial<131, 7> { using ZPZ = aerobus::zpz<131>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<129»; };
                                       template<> struct ConwayPolynomial<131, 8> { using ZPZ = aerobus::zpz<131>, using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<7>, ZPZV<116>, ZPZV<104>, ZPZV<2»; }; //
                        NOLINT
                                       template<> struct ConwayPolynomial<131, 9> { using ZPZ = aerobus::zpz<131>; using type =
03920
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<19>, ZPZV<129»; };
03921
                                         template<> struct ConwayPolynomial<131, 10> { using ZPZ = aerobus::zpz<131>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<97>, ZPZV<9>, ZPZV<426>, ZPZV<44>,
                         ZPZV<2»; }; // NOLINT</pre>
03922
                                       template<> struct ConwavPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<6>, ZPZV<129»; };</pre>
                                                                                                                          // NOLINT
                                        template<> struct ConwayPolynomial<131, 12> { using ZPZ = aerobus::zpz<131>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<50>, ZPZV<122>, ZPZV<40>, ZPZV<40>, ZPZV<125>,
                         \mbox{ZPZV}\mbox{<}28\mbox{>}, \mbox{ZPZV}\mbox{<}103\mbox{>}, \mbox{ZPZV}\mbox{<}2\mbox{*}; \mbox{}\}; \mbox{}//\mbox{}NOLINT
                                         template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type =
03924
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
 03925
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                  // NOLINT
                         template<> struct ConwayPolynomial<131, 19> { using ZPZ = aerobus::zpz<131>; using type =
03926
                         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<9>, ZPZV<9>, ZPZV<129»; }; //</pre>
                                       template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
                        POLYV<ZPZV<1>, ZPZV<134»; }; // NOLINT
 03928
                                         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 =
03929
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
 03930
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<95>, ZPZV<3»; }; // NOLINT
                                         template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
 03931
                        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 =
03932
                        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
 03933
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<134*; }; // NOLINT template<> struct ConwayPolynomial<137, 8> { using ZPZ = aerobus::zpz<137>; using type =
03934
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105>, ZPZV<21>, ZPZV<34>, ZPZV<34>; };
                         NOLINT
                                        template<> struct ConwayPolynomial<137, 9> { using ZPZ = aerobus::zpz<137>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<80>, ZPZV<122>, ZPZV<134»;
                         }; // NOLINT
03936
                                        template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, 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»; }; // NOLINT
                                        template<> struct ConwayPolynomial<137, 12> { using ZPZ = aerobus::zpz<137>; using type =
 03938
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<61>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<36>,
                         ZPZV<135>, ZPZV<61>, ZPZV<3»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<137, 13> { using ZPZ = aerobus::zpz<137>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<134»; }; // NOLINT</pre>
03940
                                          template<> struct ConwayPolynomial<137, 17> { using ZPZ = aerobus::zpz<137>; using type =
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
03941
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<18>, ZPZV<18*, ZPZV<134*; }; //</pre>
                                          template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
03942
                         POLYV<ZPZV<1>, ZPZV<137»; }; // NOLINT
                                           template<> struct ConwayPolynomial<139, 2> { using ZPZ = aerobus::zpz<139>; using type =
                         POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<139, 3> { using ZPZ = aerobus::zpz<139>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<137»; }; // NOLINT template<> struct ConwayPolynomial<139, 4> { using ZPZ = aerobus::zpz<139>; using type =
03945
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<139, 5> { using ZPZ = aerobus::zpz<139>; using type =
03946
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<137»; }; // NOLINT
 03947
                                         template<> struct ConwayPolynomial<139, 6> { using ZPZ = aerobus::zpz<139>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<46>, ZPZV<10>, ZPZV<118>, ZPZV<2»; }; // NOLINT
 03948
                                          template<> struct ConwayPolynomial<139, 7> { using ZPZ = aerobus::zpz<139>; using type =
                        POLYV-ZPZV<1>, ZPZV<0>, ZPZV<137»; }; // NOLINT template<> struct ConwayPolynomial<139, 8> { using ZPZ = aerobus::zpz<139>; using type =
03949
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>, ZPZV<36>, ZPZV<21>, ZPZV<2»; };
                                       template<> struct ConwayPolynomial<139, 9> { using ZPZ = aerobus::zpz<139>; using type =
03950
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<70>, ZPZV<70>, ZPZV<87>, ZPZV<137»; };
                          // NOLINT
                                          template<> struct ConwayPolynomial<139, 10> { using ZPZ = aerobus::zpz<139>; using type =
03951
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>,
                         ZPZV<106>, ZPZV<2»; };</pre>
                                                                                                                            // NOLINT
                                         template<> struct ConwayPolynomial<139, 11> { using ZPZ = aerobus::zpz<139>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<7>, ZPZV<137>; // NOLINT
                         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<77>, ZPZV<70>, ZPZV<106>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
03953
03954
                                        template<> struct ConwayPolynomial<139, 13> { using ZPZ = aerobus::zpz<139>; using type :
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<137»; }; // NOLINT
template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type =
03955
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<137»; };</pre>
                                        template<> struct ConwayPolynomial<139, 19> { using ZPZ = aerobus::zpz<139>; using type
 03956
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<137»; }; //</pre>
                         NOLINT
03957
                                          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 =
                         POLYV<ZPZV<1>, ZPZV<145>, ZPZV<2»; }; // NOLINT
 03959
                                          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 =
03960
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<2»; };
                                                                                                                                                                                                                                                                   // NOLINT
                                           template<> struct ConwayPolynomial<149, 5> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<147»; }; // NOLINT
 03962
                                        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»; }; // NOLINT
                                        template<> struct ConwayPolynomial<149, 7> { using ZPZ = aerobus::zpz<149>; using type =
03963
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<19>, ZPZV<147»; }; // NOLINT
                                           template<> struct ConwayPolynomial<149, 8> { using ZPZ = aerobus::zpz<149>; using type
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<140>, ZPZV<25>, ZPZV<123>, ZPZV<123>; };
                         \label{eq:convayPolynomial<149, 9> { using ZPZ = $aerobus::zpz<149>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<146>, ZPZV<20>, ZPZV<147*;}
03965
                         }; // NOLINT
   template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<74>, 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<ZPZV<13>, ZPZV<0>, ZPZV<
                                           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<91>, ZPZV<52>, ZPZV<9>,
                         ZPZV<104>, ZPZV<110>, ZPZV<2»; };  // NOLINT</pre>
                                        template<> struct ConwayPolynomial<149, 13> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
```

```
template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             // NOLINT
                            template<> struct ConwayPolynomial</ri>

149, 19> { using ZPZ = aerobus::zpz<149>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
03971
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<147»; }; //</pre>
                             NOLINT
03972
                                                template<> struct ConwayPolynomial<151, 1> { using ZPZ = aerobus::zpz<151>; using type
                            POLYV<ZPZV<1>, ZPZV<145»; }; // NOLINT
                                              template<> struct ConwayPolynomial<151, 2> { using ZPZ = aerobus::zpz<151>; using type =
03973
                            POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6»; }; // NOLINT
                                              template<> struct ConwayPolynomial<151, 3> { using ZPZ = aerobus::zpz<151>; using type =
03974
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<145»; }; // NOLINT template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
                            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<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<145>; }; // NOLINT
03976
03977
                            template<> struct ConwayPolynomial<151, 6> { using ZPZ = aerobus::zpz<151>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<125>, ZPZV<18>, ZPZV<15>, ZPZV<6»; }; // NOLINT
                                              template<> struct ConwayPolynomial<151,
                                                                                                                                                                                                                                            7> { using ZPZ = aerobus::zpz<151>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<145»; }; //
03979
                                              template<> struct ConwayPolynomial<151, 8> { using ZPZ = aerobus::zpz<151>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<140>, ZPZV<122>, ZPZV<43>, ZPZV<6»; }; //
                            NOLINT
03980
                                               template<> struct ConwayPolynomial<151, 9> { using ZPZ = aerobus::zpz<151>; using type
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<126>, ZPZV<96>, ZPZV<145»;
                             }; // NOLINT
03981
                                             template<> struct ConwayPolynomial<151, 10> { using ZPZ = aerobus::zpz<151>; using type
                             \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 104>, \ \texttt{ZPZV} < 49>, \ \texttt{ZPZV} < 20^>, \ \texttt{ZPZV} < 142>, \ \texttt{ZPZV} < 104>, \ \texttt{ZPZV} < 1
                             ZPZV<6»; }; // NOLINT
                                              template<> struct ConwayPolynomial<151, 11> { using ZPZ = aerobus::zpz<151>; using type =
03982
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<1>, ZPZV<145»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<151, 12> { using ZPZ = aerobus::zpz<151>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<121>, ZPZV<101>, ZPZV<101>, ZPZV<6>, ZPZV<77>,
                             ZPZV<107>, ZPZV<147>, ZPZV<6»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<151, 13> { using ZPZ = aerobus::zpz<151>; using type =
03984
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<145»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<151,</pre>
                                                                                                                                                                                                                                            17> { using ZPZ = aerobus::zpz<151>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<145»; }; // NOLINT
    template<> struct ConwayPolynomial<151, 19> { using ZPZ = aerobus::zpz<151>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03986
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9</pre>, ZPZV<145»; }; //
                             NOLINT
03987
                                              template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
                            POLYV<ZPZV<1>, ZPZV<152»; }; // NOLINT
                                               template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
03988
                            POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5»; }; // NOLINT
                                               template<> struct ConwayPolynomial<157, 3> { using ZPZ = aerobus::zpz<157>; using type =
03989
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<152»; }; // NOLINT
                                              template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
03990
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<136>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<157, 5> { using ZPZ = aerobus::zpz<157>; using type =
03991
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<152»; }; // NOLINT template<> struct ConwayPolynomial<157, 6> { using ZPZ = aerobus::zpz<157>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<130>, ZPZV<43>, ZPZV<144>, ZPZV<5»; }; // NOLINT
                                               template<> struct ConwayPolynomial<157, 7> { using ZPZ = aerobus::zpz<157>; using type =
03993
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<14>, ZPZV<152»; }; // NoLII template<> struct ConwayPolynomial<157, 8> { using ZPZ = aerobus::zpz<157>; using type =
03994
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<97>, ZPZV<40>, ZPZV<153>, ZPZV<5»; }; //
                            NOLINT
                                              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
03996
                                                template<> struct ConwayPolynomial<157, 10> { using ZPZ = aerobus::zpz<157>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<61>, ZPZV<22>, ZPZV<124>, ZPZV<61>, ZPZV<93>,
                             ZPZV<5»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<157, 11> { using ZPZ = aerobus::zpz<157>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<29>, ZPZV<152»; }; // NOLINT</pre>
                            \label{eq:convayPolynomial} $$\operatorname{PZPZ} = \operatorname{aerobus::zpz<157}; \text{ using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110>, ZPZV<72>, ZPZV<137>, ZPZV<43>, ZPZV<10>, ZPZV<
03998
                             ZPZV<152>, ZPZV<57>, ZPZV<5»; }; // NOLINT</pre>
                                                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<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                                template<> struct ConwayPolynomial<157, 19> { using ZPZ = aerobus::zpz<157>, using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; }; //</pre>
                                              template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
04002
                             POLYV<ZPZV<1>, ZPZV<161»; }; // NOLINT
```

```
04003
                                                  template<> struct ConwayPolynomial<163, 2> { using ZPZ = aerobus::zpz<163>; using type =
                             POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; }; // NOLINT
04004
                                               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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<163, 5> { using ZPZ = aerobus::zpz<163>; using type =
 04005
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<161»; }; // NOLINT
 04007
                                                template<> struct ConwayPolynomial<163, 6> { using ZPZ = aerobus::zpz<163>; using type =
                              \texttt{POLYV} < \texttt{ZPZV} < 1 >, \ \texttt{ZPZV} < 0 >, \ \texttt{ZPZV} < 0 >, \ \texttt{ZPZV} < 28 >, \ \texttt{ZPZV} < 25 >, \ \texttt{ZPZV} < 156 >, \ \texttt{ZPZV} < 2 >; \ \}; \ \ // \ \ \texttt{NOLINT} 
                                                template<> struct ConwayPolynomial<163, 7> { using ZPZ = aerobus::zpz<163>; using type =
04008
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<161»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<163, 8> { using ZPZ = aerobus::zpz<163>; using type =
04009
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<132>, ZPZV<83>, ZPZV<6>, ZPZV<2»; };
                              NOLINT
04010
                                                template<> struct ConwayPolynomial<163, 9> { using ZPZ = aerobus::zpz<163>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<162>, ZPZV<162>, ZPZV<161»;
                              }; // NOLINT
                                                  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>
04012
                                                  template<> struct ConwayPolynomial<163, 11> { using ZPZ = aerobus::zpz<163>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04013
                                                 template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type =
                             POLYY<ZPZV<10>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>, ZPZV<103>, ZPZV<103>, ZPZV<10>, ZPZV<30>, ZPZV<103>, ZPZV<10>, ZPZV<30>, ZPZV<103>, ZPZV<30>, ZPZV<30>,
04014
                                               template<> struct ConwayPolynomial<163, 13> { using ZPZ = aerobus::zpz<163>; using type =
                              \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                             ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<161»; }; // NOLINT
template<> struct ConwayPolynomial<163, 17> { 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<0>, ZPZV<71>, ZPZV<161»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<163, 19> { using ZPZ = aerobus::zpz<163>; using type =
                                                                                                                                                                                                                                                                                                                                                                                                                                                                  ZPZV<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<161»; }; //</pre>
                              NOLINT
                                                 template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
                             POLYV<ZPZV<1>, ZPZV<162»; }; // NOLINT
                                                template<> struct ConwayPolynomial<167, 2> { using ZPZ = aerobus::zpz<167>; using type =
                            POLYV<ZPZV<1>, ZPZV<166>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<167, 3> { using ZPZ = aerobus::zpz<167>; using type =
04019
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162»; }; // NOLINT template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type =
04020
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; }; // NOLINT
 04021
                                                 template<> struct ConwayPolynomial<167, 5> { using ZPZ = aerobus::zpz<167>; using type =
                             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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<75>, ZPZV<38>, ZPZV<2>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 7> { using ZPZ = aerobus::zpz<167>; using type =
04022
04023
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162»; }; //
                                                template<> struct ConwayPolynomial<167, 8> { using ZPZ = aerobus::zpz<167>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<149>, ZPZV<56>, ZPZV<113>, ZPZV<5»; };
04025
                                                template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<165>, ZPZV<122>, ZPZV<162»;
                                                  template<> struct ConwayPolynomial<167, 10> { using ZPZ = aerobus::zpz<167>; using type =
 04026
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<60>, ZPZV<6>, ZPZV<60>, Z
                             ZPZV<148>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type =
04027
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<24>, ZPZV<162»; }; // NOLINT</pre>
                                                  template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type
                              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, 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
04029
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162»; }; // NOLINT</pre>
                                                  template<> struct ConwayPolynomial<167,
                                                                                                                                                                                                                                                  17> { using ZPZ = aerobus::zpz<167>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<162»; }; // NOLINT
template<> struct ConwayPolynomial<167, 19> { using ZPZ = aerobus::zpz<167>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04031
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<162»; }; //</pre>
 04032
                                                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 =
04033
                             POLYV<ZPZV<1>, ZPZV<169>, ZPZV<2»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
 04034
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<2»; }; // NOLINT
 04036
                                              template<> struct ConwayPolynomial<173, 5> { using ZPZ = aerobus::zpz<173>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171»; }; // NOLINT
                                                template<> struct ConwayPolynomial<173, 6> { using ZPZ = aerobus::zpz<173>; using type =
 04037
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<134>, ZPZV<107>, ZPZV<2»; };
                       template<> struct ConwayPolynomial<173, 7> { using ZPZ = aerobus::zpz<173>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<171»; }; // NOLINT
                                     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<2*; }; //
                       NOT.TNT
                                       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<0>, 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<164>, ZPZV<48>, ZPZV<106>,
04041
                       ZPZV<58>, ZPZV<2»; }; // NOLINT
   template<> struct ConwayPolynomial<173, 11> { using ZPZ = aerobus::zpz<173>; using type
04042
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<12>, ZPZV<171»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type =
04043
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<26>, ZPZV<46>, ZPZV<46>, ZPZV<166>, ZPZV<0>,
                       ZPZV<159>, ZPZV<22>, ZPZV<22>; }; // NOLINT template<> struct ConwayPolynomial<173, 13> { using ZPZ = aerobus::zpz<173>; 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<6>, ZPZV<171»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type :
04045
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<171*); }; // NOLINT template<> struct ConwayPolynomial<173, 19> { using ZPZ = aerobus::zpz<173>; using type =
04046
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>
                        NOLINT
04047
                                        template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
                       POLYV<ZPZV<1>, ZPZV<177»; }; // NOLINT
                                      template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
04048
                       POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<179, 3> { using ZPZ = aerobus::zpz<179>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<177»; }; // NOLINT
                                        template<> struct ConwayPolynomial<179, 4> { using ZPZ = aerobus::zpz<179>; using type =
 04050
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<2»; }; // NOLINT
                                       template<> struct ConwayPolynomial<179, 5> { using ZPZ = aerobus::zpz<179>; using type =
04051
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<177»; }; // NOLINT
 04052
                                        template<> struct ConwayPolynomial<179, 6> { using ZPZ = aerobus::zpz<179>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<9>, ZPZV<5>, ZPZV<55>, ZPZV<109>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
 04053
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>; ZPZV<6>; ZPZV<6>; zPZV<6>; zPZV<6>; zPZV<6>; zPZV<6 ; zPZV<6
04054
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<144>, ZPZV<144>, ZPZV<73>, ZPZV<2»; }; //
 04055
                                       template<> struct ConwayPolynomial<179, 9> { using ZPZ = aerobus::zpz<179>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<40>, ZPZV<64>, ZPZV<177»; };
                        // NOLINT
                                       template<> struct ConwayPolynomial<179, 10> { using ZPZ = aerobus::zpz<179>; using type =
04056
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<115>, ZPZV<71>, ZPZV<150>, ZPZV<49>, ZPZV<87>,
                        ZPZV<2»; }; // NOLINT
                                      template<> struct ConwayPolynomial<179, 11> { using ZPZ = aerobus::zpz<179>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<28>, ZPZV<177»; }; // NOLINT
04058
                                      template<> struct ConwayPolynomial<179, 12> { using ZPZ = aerobus::zpz<179>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<103>, ZPZV<83>, ZPZV<43>, ZPZV<76>, ZPZV<8>, ZPZV<177>, ZPZV<1>, ZPZV<1>, ZPZV<2>; }; // NOLINT
                                     template<> struct ConwayPolynomial<179, 13> { using ZPZ = aerobus::zpz<179>; using type =
 04059
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<177»; }; // NOLINT
template<> struct ConwayPolynomial<179, 17> { using ZPZ = aerobus::zpz<179>; using type =
04060
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<177»; }; // NOLINT template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<11>, ZPZV<177»; }; //</pre>
                        NOLINT
04062
                                       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 =
                        POLYV<ZPZV<1>, ZPZV<177>, ZPZV<2»; }; // NOLINT
 04064
                                     template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179»; }; // NOLINT template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
 04065
                       POLYV<ZPZV<1>, ZPZV<6>, ZPZV<65, ZPZV<105>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<179»; }; // NOLINT
                                     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 =
04068
                       POLYV<ZPZV<1>, 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 =
 04069
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<108>, ZPZV<22>, ZPZV<149>, ZPZV<2*; };
                        NOLINT
04070
                                     template<> struct ConwayPolynomial<181, 9> { using ZPZ = aerobus::zpz<181>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<107>, ZPZV<168>, ZPZV<179»;
                        }; // NOLINT
```

```
template<> struct ConwayPolynomial<181, 10> { using ZPZ = aerobus::zpz<181>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<154>, ZPZV<104>, ZPZV<94>, ZPZV<57>, ZPZV<88>,
                         ZPZV<2»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<181, 11> { using ZPZ = aerobus::zpz<181>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04072
                                                                                                                                // NOLINT
                         ZPZV<24>, ZPZV<179»; };
                                          template<> struct ConwayPolynomial<181, 12> { using ZPZ = aerobus::zpz<181>; using type =
                         POLYV<ZPZV<1>, 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 =
04074
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         template<> struct ConwayPolynomial<181, 17> { using ZPZ = aerobus::zpz<181>; using type
04075
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<179»; };</pre>
                                                                                                                                                                                                                                                                                                                                                             // NOLINT
                        template<> struct ConwayPolynomial<181, 19> { using ZPZ = aerobus::zpz<181>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV
04076
                                          template<> struct ConwayPolynomial<191, 1> { using ZPZ = aerobus::zpz<191>; using type =
                        POLYV<ZPZV<1>, ZPZV<172»; }; // NOLINT
                                         template<> struct ConwayPolynomial<191, 2> { using ZPZ = aerobus::zpz<191>; using type =
04078
                       POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19»; }; // NOLINT
template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
04079
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172»; }; // NOLINT
                                          template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; }; // NOLINT
04081
                                         template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<172»; }; // NOLINT
                       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 =
04082
04083
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<14>, ZPZV<172»; };
                                      template<> struct ConwayPolynomial<191, 8> { using ZPZ = aerobus::zpz<191>; using type =
04084
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<164>, ZPZV<139>, ZPZV<171>, ZPZV<19»; }; //
                         NOLINT
                                        template<> struct ConwayPolynomial<191, 9> { using ZPZ = aerobus::zpz<191>; using type
04085
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<62>, ZPZV<124>, ZPZV<172»;
                         }; // NOLINT
                                          template<> struct ConwayPolynomial<191, 10> { using ZPZ = aerobus::zpz<191>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<113>, ZPZV<47>, ZPZV<173>, ZPZV<74>,
                        ZPZV<156>, ZPZV<19»; }; // NOLINT
  template<> struct ConwayPolynomial<191, 11> { using ZPZ = aerobus::zpz<191>; using type :
04087
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<6>, ZPZV<172»; }; // NOLINT</pre>
04088
                                         template<> struct ConwayPolynomial<191, 12> { using ZPZ = aerobus::zpz<191>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<168>, ZPZV<25>, ZPZV<49>, ZPZV<90>,
                         ZPZV<7>, ZPZV<151>, ZPZV<19»; }; // NOLINT</pre>
04089
                                        template<> struct ConwayPolynomial<191, 13> { using ZPZ = aerobus::zpz<191>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<172»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<191, 17> { using ZPZ = aerobus::zpz<191>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<172»; }; // NOLINT
template<> struct ConwayPolynomial<191, 19> { using ZPZ = aerobus::zpz<191>; using type :
04091
                        POLYVCZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<3 , ZPZV<4 , ZPZV<4
                                          template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
04092
                        POLYV<ZPZV<1>, ZPZV<188»; }; // NOLINT
                                        template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
04093
                        POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5»; }; // NOLINT
04094
                                          template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
                        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
04095
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<5»; }; // NOLINT
                                         template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
04096
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<188»; }; // NOLINT
                                         template<> struct ConwayPolynomial<193, 6> { using ZPZ = aerobus::zpz<193>; using type =
04097
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<8>, ZPZV<172>, ZPZV<5»; };
                                         template<> struct ConwayPolynomial<193, 7> { using ZPZ = aerobus::zpz<193>; using type =
04098
                        POLYV<ZPZV<1>, ZPZV<0>, 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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<145>, ZPZV<34>, ZPZV<154>, ZPZV<154>, ZPZV<5»; }; //</pre>
04099
                         NOLINT
                                         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<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<20>, ZPZV<20>, ZPZV<51>, ZPZV<77>, ZPZV<77>, ZPZV<89>,
                         ZPZV<5»; }; // NOLINT
                                          template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<1>, ZPZV<188»; }; // NOLINT</pre>
04103
                                        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<155>,
                          ZPZV<90>, ZPZV<46>, ZPZV<28>, ZPZV<5»; }; // NOLINT</pre>
```

```
template<> struct ConwayPolynomial<193, 13> { using ZPZ = aerobus::zpz<193>; using type :
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<188»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<193, 17> { using ZPZ = aerobus::zpz<193>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<188»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                                                         // NOLINT
                                              template<> struct ConwayPolynomial<193, 19> { using ZPZ = aerobus::zpz<193>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<188»; }; //</pre>
                            NOLINT
04107
                                              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 =
04108
                           POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2»; }; // NOLINT
04109
                                            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 =
04110
                          POLYVCZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<124>, ZPZV<2; ); // NOLINT template<> struct ConwayPolynomial<197, 5> { using ZPZ = aerobus::zpz<197>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195»; }; // NOLINT
                                               template<> struct ConwayPolynomial<197, 6> { using ZPZ = aerobus::zpz<197>; using type =
04112
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<124>, ZPZV<79>, ZPZV<173>, ZPZV<2»; }; // NOLINT
                                            template<> struct ConwayPolynomial<197, 7> { using ZPZ = aerobus::zpz<197>; using type =
04113
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04114
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<176>, ZPZV<96>, ZPZV<29>, ZPZV<29; };
04115
                                            template<> struct ConwayPolynomial<197, 9> { using ZPZ = aerobus::zpz<197>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<127>, ZPZV<8>, ZPZV<195»;
                            }; // NOLINT
                                              template<> struct ConwayPolynomial<197, 10> { using ZPZ = aerobus::zpz<197>; using type :
04116
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<121>, ZPZV<137>, ZPZV<8>, ZPZV<73>, ZPZV<42>,
                             ZPZV<2»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<197, 11> { using ZPZ = aerobus::zpz<197>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<14>, ZPZV<195»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<197, 12> { using ZPZ = aerobus::zpz<197>; using type =
04118
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>,
                            ZPZV<90>, ZPZV<163>, ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<197, 13> { using ZPZ = aerobus::zpz<197>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04120
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            04121
                                            template<> struct ConwayPolynomial<197, 19> { using ZPZ = aerobus::zpz<197>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0</pre>
                            NOLINT
04122
                                              template<> struct ConwavPolynomial<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 =
                           POLYV<ZPZV<1>, ZPZV<193>, ZPZV<3»; }; // NOLINT
04124
                                              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 =
04125
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; };
                                                                                                                                                                                                                                                                                                   // NOLINT
                                            template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
04126
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
04127
                                               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
04128
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; };
                                              template<> struct ConwayPolynomial<199, 8> { using ZPZ = aerobus::zpz<199>; using type
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<160>, ZPZV<23>, ZPZV<159>, ZPZV<3»; }; //
                           NOLINT
04130
                                              template<> struct ConwayPolynomial<199, 9> { using ZPZ = aerobus::zpz<199>; using type
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<141>, ZPZV<196»;
04131
                                                template<> struct ConwayPolynomial<199, 10> { using ZPZ = aerobus::zpz<199>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<158>, ZPZV<31>, ZPZV<54>, ZPZV<9>,
                            ZPZV<3»; }; // NOLINT</pre>
                           \label{eq:convayPolynomial} $$ \ensuremath{\texttt{LPS}}, 11> \{ using \ensuremath{\texttt{ZPZ}} = aerobus:: zpz<199>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZP
04132
                                                                                                                                           // NOLINT
                            ZPZV<1>, ZPZV<196»; };</pre>
                                               template<> struct ConwayPolynomial<199, 12> { using ZPZ = aerobus::zpz<199>; using type =
                            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
                                               template<> struct ConwayPolynomial<199,
                                                                                                                                                                                                                                        17> { using ZPZ = aerobus::zpz<199>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             \texttt{ZPZV} < \texttt{0>, ZPZV} < \texttt{13>, ZPZV} < \texttt{196} *; }; \\
                                                                                                                                                                                                                                                                                                                                                                                                       // NOLINT
                           template<> struct ConwayPolynomial<199, 19> { using ZPZ = aerobus::zpz<199>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<19>, ZPZV<19</p>
04136
```

```
NOLINT
                                       template<> struct ConwayPolynomial<211, 1> { using ZPZ = aerobus::zpz<211>; using type =
                      POLYV<ZPZV<1>, ZPZV<209»; }; // NOLINT template<> struct ConwayPolynomial<211, 2> { using ZPZ = aerobus::zpz<211>; using type =
 04138
                      POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; }; // NOLINT
                                       template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
04139
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<209»; }; // NOLINT
                                       template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
 04140
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<211, 5> { using ZPZ = aerobus::zpz<211>; using type =
 04141
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2), ZPZV<1>, ZPZV<20»; }; // NOLINT template<> struct ConwayPolynomial<211, 6> { using ZPZ = aerobus::zpz<211>; using type =
04142
                      POLYV<2PZV<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 =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<209»; }; // NOLINT
 04144
                                     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<2»; }; //
                       NOLINT
                                       template<> struct ConwayPolynomial<211, 9> { using ZPZ = aerobus::zpz<211>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<139>, ZPZV<139>, ZPZV<26>, ZPZV<209»;
                       }; // NOLINT
04146
                                      template<> struct ConwayPolynomial<211, 10> { using ZPZ = aerobus::zpz<211>; using type :
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<5>, ZPZV<61>, ZPZV<148>, ZPZV<148>, ZPZV<125>,
                        ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<211, 11> { using ZPZ = aerobus::zpz<211>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                    // NOLINT
                        ZPZV<7>, ZPZV<209»; };</pre>
04148
                                    template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type :
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<50, ZPZV<145>, ZPZV<126>, ZPZV<184>,
                        ZPZV<84>, ZPZV<27>, ZPZV<2»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<211, 13> { using ZPZ = aerobus::zpz<211>; using type
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<209»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<211, 17> { using ZPZ = aerobus::zpz<211>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       template<> struct ConwayPolynomial<211, 19> { using ZPZ = aerobus::zpz<211>; using type =
04151
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<209»; }; //</pre>
                                      template<> struct ConwayPolynomial<223, 1> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYV<ZPZV<1>, ZPZV<220»; }; // NOLINT
                                       template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
04153
                      POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; }; // NOLINT
                                        template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
 04155
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<63, ZPZV<3>, ; // NOLINT template<> struct ConwayPolynomial<223, 5> { using ZPZ = aerobus::zpz<223>; using type =
04156
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; }; // NOLINT
04157
                                       template<> struct ConwayPolynomial<223, 6> { using ZPZ = aerobus::zpz<223>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68>, ZPZV<24>, ZPZV<196>, ZPZV<3»; }; // NOLINT
 04158
                                    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<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
04159
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<138>, ZPZV<3»; }; //
                                    template<> struct ConwayPolynomial<223, 9> { using ZPZ = aerobus::zpz<223>; using type =
 04160
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<164>, ZPZV<64>, ZPZV<220»;
                        }; // NOLINT
04161
                                       template<> struct ConwayPolynomial<223, 10> { using ZPZ = aerobus::zpz<223>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<118>, ZPZV<177>, ZPZV<87>, ZPZV<99>, ZPZV<62>,
                        ZPZV<3»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<223, 11> { using ZPZ = aerobus::zpz<223>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<8>, ZPZV<220»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<223, 12> { using ZPZ = aerobus::zpz<223>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<64>, ZPZV<94>, ZPZV<11>, ZPZV<105>, ZPZV<64>, ZPZV<151>, ZPZV<213>, ZPZV<3»; }; // NOLINT
04163
                                        template<> struct ConwayPolynomial<223, 13> { using ZPZ = aerobus::zpz<223>; using type
                        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
04165
                       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
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<20»; }; //</pre>
                        NOLINT
04167
                                      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 =
                      POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2»; }; // NOLINT
 04169
                                    template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<225»; }; // NOLINT
template<> struct ConwayPolynomial<227, 4> { using ZPZ = aerobus::zpz<227>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<227, 5> { using ZPZ = aerobus::zpz<227>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<225»; }; // 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<29; }; // NOLINT template<> struct ConwayPolynomial<227, 7> { using ZPZ = aerobus::zpz<227>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>; }; // NOL
                                    template<> struct ConwayPolynomial<227, 8> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<151>, ZPZV<176>, ZPZV<106>, ZPZV<2»; }; //
04175
                                  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<2>, ZPZV<2>, ZPZV<2+, ZPZV<24>, ZPZV<24>, ZPZV<183>, ZPZV<225»;
                      }; // NOLINT
04176
                                      template<> struct ConwayPolynomial<227, 10> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>, ZPZV<12>, ZPZV<12>, ZPZV<93>, ZPZV<77>,
                      ZPZV<2»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type =
04177
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<2>, ZPZV<225»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<123>, ZPZV<99>, ZPZV<160>, ZPZV<96>, ZPZV<127>, ZPZV<142>, ZPZV<94>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<227, 13> { using ZPZ = aerobus::zpz<227>; using type =
04179
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<225»; }; // NOLINT template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<225»; };</pre>
                                  template<> struct ConwayPolynomial<227, 19> { using ZPZ = aerobus::zpz<227>; using type =
04181
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      NOLINT
04182
                                      template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<223»; }; // NOLINT
                                  template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
                     POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
 04184
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; ); // NOLINT template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<6»; }; // NOLINT
                                   template<> struct ConwayPolynomial<229, 5> { using ZPZ = aerobus::zpz<229>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223»; }; // NOLINT
                     template<> struct ConwayPolynomial<229, 6> { using ZPZ = aerobus::zpz<229>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<160>, ZPZV<186>, ZPZV<6»; }; // NOLINT
04187
                                    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»; }; //
 04189
                                    template<> struct ConwayPolynomial<229, 8> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<193>, ZPZV<62>, ZPZV<205>, ZPZV<6»; }; //
04190
                                    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<15>, ZPZV<117>, ZPZV<50>, ZPZV<223»;
                     }; // NOLINT template<> struct ConwayPolynomial<229, 10> { using ZPZ = aerobus::zpz<229>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<185>, ZPZV<135>, ZPZV<158>, ZPZV<167>,
                      ZPZV<98>, ZPZV<6»; };</pre>
                                                                                                         // NOLINT
                                    template<> struct ConwayPolynomial<229, 11> { using ZPZ = aerobus::zpz<229>; using type =
04192
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  template<> struct ConwayPolynomial<229, 12> { using ZPZ = aerobus::zpz<229>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<131>, ZPZV<140>, ZPZV<25>, ZPZV<6>, ZPZV<172>, ZPZV<9>, ZPZV<445>, ZPZV<6»; }; // NOLINT
                                    template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type =
04194
                      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 =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 2>, \ \texttt{ZPZV} < 2> , \ \texttt{ZPZV} < 2> 3 ; \ // \ \texttt{NOLINT} 
                      template<> struct ConwayPolynomial<229, 19> { using ZPZ = aerobus::zpz<229>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
04196
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<228>, ZPZV<15>, ZPZV<223»; }; //</pre>
                      NOLINT
                                      template<> struct ConwayPolynomial<233, 1> { using ZPZ = aerobus::zpz<233>; using type =
                     POLYV<ZPZV<1>, ZPZV<230»; }; // NOLINT
                                     template<> struct ConwayPolynomial<233, 2> { using ZPZ = aerobus::zpz<233>; using type =
04198
                      POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
                                    template<> struct ConwayPolynomial2233, 3> { using ZPZ = aerobus::zpz<233>; using type =
04199
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<230»; }; // NOLINT
                                   template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
 04200
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<4+, ZPZV<158>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<233, 5> { using ZPZ = aerobus::zpz<233>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<230»; }; // NOLINT
template<> struct ConwayPolynomial<233, 6> { using ZPZ = aerobus::zpz<233>; using type =
 04201
 04202
                     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 =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230»; }; // NOLINT
04204
                                  template<> struct ConwayPolynomial<233, 8> { using ZPZ = aerobus::zpz<233>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<181>, ZPZV<181>, ZPZV<3»; }; //
```

```
template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<146>, ZPZV<230»;
                          }; // NOLINT
                         template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<28>, ZPZV<71>, ZPZV<102>, ZPZV<3>, ZPZV<48>,
 04206
                          ZPZV<3»: }: // NOLINT
                                           template<> struct ConwayPolynomial<233, 11> { using ZPZ = aerobus::zpz<233>; using type
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<5>, ZPZV<230»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<233, 12> { using ZPZ = aerobus::zpz<233>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<114>, ZPZV<31>, ZPZV<31>, ZPZV<19>,
04208
                          ZPZV<216>, ZPZV<20>, ZPZV<3»: };  // NOLINT</pre>
                                          template<> struct ConwayPolynomial<233, 13> { using ZPZ = aerobus::zpz<233>; using type
04209
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<230»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<233, 17> { using ZPZ = aerobus::zpz<233>; using type =
04210
                          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<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 19> { using ZPZ = aerobus::zpz<233>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<25>, ZPZV<230»; }; //</pre>
                          NOLINT
04212
                                         template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
                         POLYV<ZPZV<1>, ZPZV<232»; }; // NOLINT
04213
                                           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 =
04214
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<232»; }; // NOLINT
                                           template<> struct ConwayPolynomial<239, 4> { using ZPZ = aerobus::zpz<239>; using type =
 04215
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<239, 5> { using ZPZ = aerobus::zpz<239>; using type =
04216
                         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 =
 04217
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<237>, ZPZV<60>, ZPZV<200>, ZPZV<7»; }; // NOLINT
 04218
                                          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<17>, ZPZV<232»; }; // NOLINT
                                         template<> struct ConwayPolynomial<239, 8> { using ZPZ = aerobus::zpz<239>; using type =
04219
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<202>, ZPZV<54>, ZPZV<7»; };
                          NOLINT
                                         template<> struct ConwayPolynomial<239, 9> { using ZPZ = aerobus::zpz<239>; using type =
04220
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<88>, ZPZV<232»; };
                          // NOLINT
                                         template<> struct ConwayPolynomial<239, 10> { using ZPZ = aerobus::zpz<239>; using type =
04221
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<68>, ZPZV<226>, ZPZV<127>, ZPZV<108>, ZPZV<7»; }; // NOLINT
                                          template<> struct ConwayPolynomial<239, 11> { using ZPZ = aerobus::zpz<239>; using type :
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<8>, ZPZV<232»; }; // NOLINT</pre>
04223
                                         template<> struct ConwayPolynomial<239, 12> { using ZPZ = aerobus::zpz<239>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<14>, 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 =
                          \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ 
                         ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; }; // NOLINT
   template<> struct ConwayPolynomial<239, 17> { using ZPZ = aerobus::zpz<239>; using type =
04225
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<232»; };</pre>
                                        template<> struct ConwayPolynomial<239, 19> { using ZPZ = aerobus::zpz<239>; using type
 04226
                          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>
                          NOLINT
                                         template<> struct ConwayPolynomial<241, 1> { using ZPZ = aerobus::zpz<241>; using type =
04227
                         POLYV<ZPZV<1>, ZPZV<234»; }; // NOLINT
                                           template<> struct ConwayPolynomial<241, 2> { using ZPZ = aerobus::zpz<241>; using type =
                         POLYV<ZPZV<1>, ZPZV<238>, ZPZV<7»; }; // NOLINT
                                         template<> struct ConwayPolynomial<241, 3> { using ZPZ = aerobus::zpz<241>; using type =
 04229
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<234»; }; // NOLINT template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
04230
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<152>, ZPZV<7»; }; // NOLINT
                                           template<> struct ConwayPolynomial<241, 5> { using ZPZ = aerobus::zpz<241>; using type =
 04231
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<234»; }; // NOLINT
 04232
                                        template<> struct ConwayPolynomial<241, 6> { using ZPZ = aerobus::zpz<241>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<6>, ZPZV<5>, ZPZV<7»; }; // NOLINT
                                          template<> struct ConwayPolynomial<241, 7> { using ZPZ = aerobus::zpz<241>; using type =
04233
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<234»; }; // NOLINT
                                          template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<173>, ZPZV<212>, ZPZV<153>, ZPZV<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<0>, ZPZV<0>, ZPZV<236>, ZPZV<125>, ZPZV<234»;
                          }; // NOLINT
                                            template<> struct ConwayPolynomial<241, 10> { using ZPZ = aerobus::zpz<241>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<27>, ZPZV<145>, ZPZV<268>, ZPZV<55>,
                          ZPZV<7»; }; // NOLINT</pre>
                                        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>
                                                        template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type =
                                    \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                                   ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<23*; }; // NOLINT template<> struct ConwayPolynomial<241, 17> { using ZPZ = aerobus::zpz<241>; using type
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<234»; }; // NOLINT</pre>
04241
                                                          template<> struct ConwayPolynomial<241, 19> { using ZPZ = aerobus::zpz<241>; using type =
                                   POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                            template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
                                   POLYV<ZPZV<1>, ZPZV<245»; }; // NOLINT
 04243
                                                         template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
                                  POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; }; // NOLINT
                                                           template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
04244
                                   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
 04246
                                                           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 template<> struct ConwayPolynomial<251, 6> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<247>, ZPZV<151>, ZPZV<60*; }; // NOLINT
 04247
                                                            template<> struct ConwayPolynomial<251, 7> { using ZPZ = aerobus::zpz<251>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
 04249
                                                        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»; }; //
                                   NOLINT
                                                         template<> struct ConwayPolynomial<251, 9> { using ZPZ = aerobus::zpz<251>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<106>, ZPZV<245»;
                                                           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»; }; // NOLINT</pre>
                                                           template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type =
04252
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    ZPZV<26>, ZPZV<245»; };</pre>
                                                                                                                                                                                    // NOLINT
                                                         template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<192>, ZPZV<53>, ZPZV<20>, ZPZV<20>, ZPZV<15>,
                                   ZPZV<201>, ZPZV<232>, ZPZV<6»; }; // NOLINT
   template<> struct ConwayPolynomial<251, 13> { using ZPZ = aerobus::zpz<251>; using type :
04254
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<245»; }; // NOLINT</pre>
04255
                                                           template<> struct ConwayPolynomial<251,
                                                                                                                                                                                                                                                                                                      17> { 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<0>, ZPZV<0>, ZPZV<45»; }; // NOLINT
template<> struct ConwayPolynomial<251, 19> { using ZPZ = aerobus::zpz<251>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0</pre>
04256
                                     ZPZV<0>, ZPZV<0>
04257
                                                           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 =
04258
                                   POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3»; }; // NOLINT
                                                            template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<254»; }; // NOLINT
                                                            template<> struct ConwayPolynomial<257, 4> { using ZPZ = aerobus::zpz<257>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<187>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<257, 5> { using ZPZ = aerobus::zpz<257>; using type =
04261
                                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<254»; }; // NOLINT
 04262
                                                            template<> struct ConwayPolynomial<257, 6> { using ZPZ = aerobus::zpz<257>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<62>, ZPZV<18>, ZPZV<18>, ZPZV<138>, ZPZV<3»; }; // NOLINT
04263
                                                        template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type =
                                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<254»; }; // NOLINT template<> struct ConwayPolynomial<257, 8> { using ZPZ = aerobus::zpz<257>; using type =
 04264
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<140>, ZPZV<162>, ZPZV<162>, ZPZV<3»; }; //
                                   NOLINT
                                                            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<0>, ZPZV<25, ZPZV<2
                                    }; // 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<20>, ZPZV<2
 04266
                                    ZPZV<3»; }; // NOLINT
                                                            template<> struct ConwayPolynomial<257, 11> { using ZPZ = aerobus::zpz<257>; using type
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     ZPZV<40>, ZPZV<254»; }; // NOLINT</pre>
                                                            template<> struct ConwayPolynomial<257, 12> { using ZPZ = aerobus::zpz<257>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<25>, ZPZV<215>, ZPZV<215>, ZPZV<2173>, 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
                                   template<> struct ConwayPolynomial<257, 17> { using ZPZ = aerobus::zpz<257>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04270
```

```
template<> struct ConwayPolynomial<257, 19> { using ZPZ = aerobus::zpz<257>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<254»; }; //</pre>
               NOLINT
04272
                         template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<ZPZV<1>, ZPZV<258»; }; // NOLINT
                         template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
               POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; }; // NOLINT
04274
                        template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; }; // NOLINT template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
04275
              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
04276
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<258»; }; // NOLINT
04277
                         template<> struct ConwayPolynomial<263, 6> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<22>, ZPZV<250>, ZPZV<25>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<263, 7> { using ZPZ = aerobus::zpz<263>; using type =
04278
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<258»; }; // NOLINT
                         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»: };
04280
                        template<> struct ConwayPolynomial<263, 9> { using ZPZ = aerobus::zpz<263>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<261>, ZPZV<29>, ZPZV<258»;
               }; // NOLINT
04281
                         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»: 1: // NOLINT
                                                                           // NOLINT
               ZPZV<119>, ZPZV<5»; };</pre>
                       template<> struct ConwayPolynomial<263, 11> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<2>, ZPZV<258»: }: // NOLINT</pre>
              template<> struct ConwayPolynomial<263, 12> { using ZPZ = aerobus::zpz<263>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<174>, ZPZV<174>, ZPZV<162>, ZPZV<252>, ZPZV<47>, ZPZV<45>, ZPZV<180>, ZPZV<5»; }; // NOLINT
04283
                       template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
04284
              POLYV<ZPZV<1>, ZPZV<267»; }; // NOLINT
                         template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
04285
              POLYV<ZPZV<1>, ZPZV<268>, ZPZV<2»; }; // NOLINT
                         template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<267»; }; // NOLINT template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
04287
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2»; }; // NOLINT
                         template<> struct ConwayPolynomial<269, 5> { using ZPZ = aerobus::zpz<269>; using type =
04288
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<267»; }; // NOLINT
04289
                        template<> struct ConwayPolynomial<269, 6> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<2PZV<1>, 2PZV<0>, ZPZV<1>, ZPZV<120>, ZPZV<101>, ZPZV<206>, ZPZV<2»; }; // NOLINT
04290
                         template<> struct ConwayPolynomial<269, 7> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<267»; }; // NOLINT template<> struct ConwayPolynomial<269, 8> { using ZPZ = aerobus::zpz<269>; using type =
04291
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<220>, ZPZV<131>, ZPZV<232>, ZPZV<2x; }; //
               NOLINT
04292
                         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>,
               }; // NOLINT
04293
                         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<20>; // NOLINT
                         template<> struct ConwayPolynomial<269, 11> { using ZPZ = aerobus::zpz<269>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<20>, ZPZV<267»; }; // NOLINT</pre>
04295
                         template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<126>, ZPZV<165>, ZPZV<63>, ZPZV<215>, ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2»; }; // NOLINT
04296
                         template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<265»; }; // NOLINT
04297
                        template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; }; // NOLINT
04298
                         template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT
                        template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type =
04299
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6»; }; // NOLINT
                         template<> struct ConwayPolynomial<271, 5> { using ZPZ = aerobus::zpz<271>; using type =
04300
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT
04301
                         template<> struct ConwayPolynomial<271, 6> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<207>, ZPZV<207>, ZPZV<81>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<271, 7> { using ZPZ = aerobus::zpz<271>; using type
04302
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT
                       template<> struct ConwayPolynomial<271, 8> { using ZPZ = aerobus::zpz<271>; using type =
04303
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<19>, ZPZV<114>, ZPZV<69>, ZPZV<69; };
               NOLINT
              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<266>, ZPZV<186>, ZPZV<266>;
04304
              }; // NOLINT

template<> struct ConwayPolynomial<271, 10> { using ZPZ = aerobus::zpz<271>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<133>, ZPZV<10>, ZPZV<256>, ZPZV<74>,
                        template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type =
04306
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
```

```
ZPZV<10>, ZPZV<265»; };</pre>
                             template<> struct ConwayPolynomial<271, 12> { using ZPZ = aerobus::zpz<271>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<116>, ZPZV<210>, ZPZV<216>, ZPZV<216-, ZPZV<2
                 ZPZV<237>, ZPZV<256>, ZPZV<130>, ZPZV<6»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
04308
                 POLYV<ZPZV<1>, ZPZV<272»; }; // NOLINT
                            template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
                 POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; }; // NOLINT
04310
                           template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; }; // NOLINT template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
04311
                 POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<22>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<277, 5> { using ZPZ = aerobus::zpz<277>; using type =
04312
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<272»; }; // NOLINT
04313
                            template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<9>, ZPZV<118>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type =
04314
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<159>, ZPZV<176>, ZPZV<5»; }; //
04316
                            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
04317
                             template<> struct ConwayPolynomial<277, 10> { using ZPZ = aerobus::zpz<277>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<253>, ZPZV<237>, ZPZV<241>,
                                                                                     // NOLINT
                 ZPZV<260>, ZPZV<5»; };</pre>
04318
                           template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                 ZPZV<5>, ZPZV<272»: }: // NOLINT</pre>
                 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<180>, ZPZV<115>, ZPZV<202>, ZPZV<5»; }; // NOLINT
04319
                          template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
04320
                 POLYV<ZPZV<1>, ZPZV<278»; }; // NOLINT
                             template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
04321
                 POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; }; // NOLINT
                            template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<278»; }; // NOLINT template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
04323
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3»; }; // NOLINT
                 template<> struct ConwayPolynomial<281, 5> { using ZPZ = aerobus::zpz<281>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<278»; }; // NOLINT
04324
04325
                            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<3»; }; // NOLINT
04326
                            template<> struct ConwayPolynomial<281, 7> { using ZPZ = aerobus::zpz<281>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<29>, ZPZV<29>, ZPZV<29>, ZPZV<29>, ZPZV<218>; ]; // NOLII template<> struct ConwayPolynomial<281, 8> { using ZPZ = aerobus::zpz<281>; using type =
                                                                                                                                                                                                                                                                     // NOLINT
04327
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195>, ZPZV<279>, ZPZV<140>, ZPZV<3»; }; //
                 NOLINT
04328
                            template<> struct ConwayPolynomial<281, 9> { using ZPZ = aerobus::zpz<281>; using type
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<148>, ZPZV<70>, ZPZV<278»;
                 }; // NOLINT
04329
                            template<> struct ConwayPolynomial<281, 10> { using ZPZ = aerobus::zpz<281>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<145>, ZPZV<13>, ZPZV<138>,
                 ZPZV<191>, ZPZV<3»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<281, 11> { using ZPZ = aerobus::zpz<281>; using type
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  ZPZV<36>, ZPZV<278»; }; // NOLINT</pre>
04331
                             template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<68>, ZPZV<103>, ZPZV<116>, ZPZV<58>, ZPZV<28>, ZPZV<29>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
04332
                 POLYV<ZPZV<1>, ZPZV<280»; }; // NOLINT
04333
                           template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
                 POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3»; }; // NOLINT
04334
                            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 =
04335
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; }; // NOLINT
                            template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
04336
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
04337
                            template<> struct ConwayPolynomial<283, 6> { using ZPZ = aerobus::zpz<283>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<68>, ZPZV<73>, ZPZV<3»; }; // NOLINT
                                                                                                                                              7> { using ZPZ = aerobus::zpz<283>; using type
04338
                            template<> struct ConwayPolynomial<283,
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<280»; }; // NOLINT
                           template<> struct ConwayPolynomial<283, 8> { using ZPZ = aerobus::zpz<283>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<179>, ZPZV<32>, ZPZV<232>, ZPZV<232>, ZPZV<23»; }; //
                 NOLINT
04340
                           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<0>, ZPZV<0>, ZPZV<280»;
                }; // NOLINT
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>,
                            template<> struct ConwayPolynomial<283, 11> { using ZPZ = aerobus::zpz<283>; using type =
04342
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
```

```
ZPZV<4>, ZPZV<280»; };</pre>
               template<> struct ConwayPolynomial<283, 12> { using ZPZ = aerobus::zpz<283>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<8>, ZPZV<8>, ZPZV<86>, ZPZV<26>, ZPZV<49>,
         ZPZV<14>, ZPZV<56>, ZPZV<3»; }; // NOLINT</pre>
04344
               template<> struct ConwayPolynomial<293, 1> { using ZPZ = aerobus::zpz<293>; using type =
         POLYV<ZPZV<1>, ZPZV<291»; }; // NOLINT
               template<> struct ConwayPolynomial<293, 2> { using ZPZ = aerobus::zpz<293>; using type =
         POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<293, 3> { using ZPZ = aerobus::zpz<293>; using type =
04346
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT template<> struct ConwayPolynomial<293, 4> { using ZPZ = aerobus::zpz<293>; using type =
04347
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
04348
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
04349
               template<> struct ConwayPolynomial<293, 6> { using ZPZ = aerobus::zpz<293>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<210>, ZPZV<260>, ZPZV<20»; }; // NOLINT template<> struct ConwayPolynomial<293, 7> { using ZPZ = aerobus::zpz<293>; using type =
04350
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<291»; }; // NOLINT
               template<> struct ConwayPolynomial<293, 8> { using ZPZ = aerobus::zpz<293>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<2*; }; //
04352
               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
04353
               template<> struct ConwayPolynomial<293, 10> { using ZPZ = aerobus::zpz<293>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<28>, ZPZV<46>, ZPZV<484>, ZPZV<24>,
         ZPZV<2»; }; // NOLINT</pre>
              template<> struct ConwayPolynomial<293, 11> { using ZPZ = aerobus::zpz<293>; using type =
04354
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
         ZPZV<3>, ZPZV<291»: }: // NOLINT</pre>
         template<> struct ConwayPolynomial<293, 12> { using ZPZ = aerobus::zpz<293>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<159>, ZPZV<210>, ZPZV<125>, ZPZV<212>, ZPZV<167>, ZPZV<144>, ZPZV<157>, ZPZV<2»; }; // NOLINT
04355
              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 =
04357
         POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<302»; }; // NOLINT template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5»; }; // NOLINT
         template<> struct ConwayPolynomial</br>
template<> struct ConwayPolynomial<br/>307, 5> { using ZPZ = aerobus::zpz<307>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<302»; }; // NOLINT
04360
04361
               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
04362
               template<> struct ConwayPolynomial<307, 7> { using ZPZ = aerobus::zpz<307>; 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<302*; }; // NOLINT template<> struct ConwayPolynomial<307, 8> { using ZPZ = aerobus::zpz<307>; using type =
04363
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<283>, ZPZV<232>, ZPZV<131>, ZPZV<5»; }; //
         NOLINT
04364
               template<> struct ConwayPolynomial<307, 9> { using ZPZ = aerobus::zpz<307>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<165>, ZPZV<70>, ZPZV<302»;
         }; // NOLINT
04365
               template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
         POLYV<ZPZV<1>, ZPZV<294»; }; // NOLINT
               template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
04366
         POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; }; // NOLINT
               template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
04367
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; }; // NOLINT template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
04368
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
04369
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294»; }; // NOLINT
               template<> struct ConwayPolynomial<311, 6> { using ZPZ = aerobus::zpz<311>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<167>, ZPZV<152>, ZPZV<17»; }; // NOLINT
04371
               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<10>, ZPZV<294»; }; // NOLINT template<> struct ConwayPolynomial<311, 8> { using ZPZ = aerobus::zpz<311>; using type =
04372
         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 =
04373
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<294»;
         }; // NOLINT
04374
               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 =
         POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; }; // NOLINT
04376
               template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; }; // NOLINT template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
04377
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<239>, ZPZV<10; ; ; // NOLINT template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
04378
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<303»; }; // NOLINT
04379
              template<> struct ConwayPolynomial<313, 6> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<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<4>, ZPZV<4>, ZPZV<303»; }; // NOLINT
04380
```

```
template<> struct ConwayPolynomial<313, 8> { using ZPZ = aerobus::zpz<313>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<306>, ZPZV<99>, ZPZV<106>, ZPZV<10»; }; //
         NOLINT
        template<> struct ConwayPolynomial<313, 9> { using ZPZ = aerobus::zpz<313>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<3030>;
04382
         }; // NOLINT
04383
               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 =
04384
        POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
04385
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<315»; }; // NOLINT
               template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
04386
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2»; }; // NOLINT
04387
               template<> struct ConwayPolynomial<317, 5> { using ZPZ = aerobus::zpz<317>; using type =
        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 =
04388
        POLYY<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<25, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<5, ZPZV<5,
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<71, ZPZV<7
               template<> struct ConwayPolynomial<317, 8> { using ZPZ = aerobus::zpz<317>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<207>, ZPZV<85>, ZPZV<31>, ZPZV<2»; };
         NOLINT
        template<> struct ConwayPolynomial<317, 9> { using ZPZ = aerobus::zpz<317>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<284>, ZPZV<286>, ZPZV<315»;</pre>
04391
         }; // NOLINT
              template<> struct ConwayPolynomial<331, 1> { using ZPZ = aerobus::zpz<331>; using type =
04392
        POLYV<ZPZV<1>, ZPZV<328»; }; // NOLINT
04393
               template<> struct ConwayPolynomial<331, 2> { using ZPZ = aerobus::zpz<331>; using type =
        POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
04394
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<328»; }; // NOLINT
               template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
04395
        04396
               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
04397
04398
               template<> struct ConwayPolynomial<331,
                                                                           7> { using ZPZ = aerobus::zpz<331>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<328»; }; // NOLINT
04399
              template<> struct ConwayPolynomial<331, 8> { using ZPZ = aerobus::zpz<331>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<24>>, ZPZV<308>, ZPZV<78>, ZPZV<3»; };
         NOLINT
04400
              template<> struct ConwayPolynomial<331, 9> { using ZPZ = aerobus::zpz<331>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<194>, ZPZV<210>, ZPZV<328»;
         }; // NOLINT
04401
              template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
        POLYV<ZPZV<1>, ZPZV<327»; }; // NOLINT
04402
               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 =
04403
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT
04404
               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 =
04405
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<337, 6> { using ZPZ = aerobus::zpz<337>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<216>, ZPZV<127>, ZPZV<109>, ZPZV<10»; }; // NOLINT
               template<> struct ConwayPolynomial<337, 7> { using ZPZ = aerobus::zpz<337>; using type =
04407
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<327»; }; // NOLINT template<> struct ConwayPolynomial<337, 8> { using ZPZ = aerobus::zpz<337>; using type =
04408
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<331>, ZPZV<246>, ZPZV<251>, ZPZV<10»; }; //
         NOLINT
              template<> struct ConwayPolynomial<337, 9> { using ZPZ = aerobus::zpz<337>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<148>, ZPZV<98>, ZPZV<327»;
         }; // NOLINT
04410
               template<> struct ConwayPolynomial<347, 1> { using ZPZ = aerobus::zpz<347>; using type =
        POLYV<ZPZV<1>, ZPZV<345»; }; // NOLINT
               template<> struct ConwayPolynomial<347, 2> { using ZPZ = aerobus::zpz<347>; using type =
04411
        POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
04412
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
04413
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
04414
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<345»; }; // NOLINT
               template<> struct ConwayPolynomial<347, 6> { using ZPZ = aerobus::zpz<347>; using type =
04415
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<343>, ZPZV<26>, ZPZV<56>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
04416
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type =
04417
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<187>, ZPZV<213>, ZPZV<117>, ZPZV<2»; }; //
04418
              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<252>, ZPZV<345»;
         }; // NOLINT
04419
              template<> struct ConwayPolynomial<349, 1> { using ZPZ = aerobus::zpz<349>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<347»; };
            template<> struct ConwayPolynomial<349, 2> { using ZPZ = aerobus::zpz<349>; using type =
       POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
04421
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; }; // NOLINT template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
04422
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<349, 5> { using ZPZ = aerobus::zpz<349>; using type =
04423
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<347»; }; // NOLINT
      template<> struct ConwayPolynomial<349, 6> { using ZPZ = aerobus::zpz<349>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<135>, ZPZV<177>, ZPZV<316>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<349, 7> { using ZPZ = aerobus::zpz<349>; using type =
04424
04425
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<347»; };
            template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<328>, ZPZV<268>, ZPZV<268>, ZPZV<20*; }; //
           template<> struct ConwayPolynomial<349, 9> { using ZPZ = aerobus::zpz<349>; using type =
04427
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<290>, ZPZV<347»;
       }; // NOLINT
            template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<ZPZV<1>, ZPZV<350»; }; // NOLINT
            template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
04429
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
04430
      POLYV<ZPZV<1>, ZPZV<3>, ZPZV<35A, ZPZV<35B); // NOLINT template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<3»; }; // NOLINT
04432
            template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<350»; }; // NOLINT
      template<> struct ConwayPolynomial<353, 6> { using ZPZ = aerobus::zpz<353>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<21>, ZPZV<215>, ZPZV<226>, ZPZV<295>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<353, 7> { using ZPZ = aerobus::zpz<353>; using type =
04433
04434
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<350»; }; //
04435
           template<> struct ConwayPolynomial<353, 8> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<1>, 2PZV<182>, 2PZV<26>, 2PZV<37>, 2PZV<3»; };
       NOLINT
       template<> struct ConwayPolynomial<353, 9> { using ZPZ = aerobus::zpz<353>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<319>, ZPZV<49>, ZPZV<350»;
04436
       }; // NOLINT
            template<> struct ConwayPolynomial<359, 1> { using ZPZ = aerobus::zpz<359>; using type =
04437
       POLYV<ZPZV<1>, ZPZV<352»; }; // NOLINT
            template<> struct ConwayPolynomial<359, 2> { using ZPZ = aerobus::zpz<359>; using type =
04438
       POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7»: }: // NOLINT
            template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
04439
      POLYV<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<352,; }; // NOLINT template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
04440
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<229>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<359, 5> { using ZPZ = aerobus::zpz<359>; using type =
04441
       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 =
04443
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; }; // NOLINT
04444
           template<> struct ConwayPolynomial<359, 8> { using ZPZ = aerobus::zpz<359>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<301>, ZPZV<143>, ZPZV<271>, ZPZV<7»; }; //
       NOLINT
            template<> struct ConwayPolynomial<359, 9> { using ZPZ = aerobus::zpz<359>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<356>, ZPZV<165>, ZPZV<352»;
       }; // NOLINT
04446
            template<> struct ConwayPolynomial<367, 1> { using ZPZ = aerobus::zpz<367>; using type =
       POLYV<ZPZV<1>, ZPZV<361»; }; // NOLINT
           template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
04447
       POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6»; }; // NOLINT
            template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<361»; }; // NOLINT
template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type =
04449
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<367, 5> { using ZPZ = aerobus::zpz<367>; using type =
04450
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<361»; }; // NOLINT
04451
            template<> struct ConwayPolynomial<367, 6> { using ZPZ = aerobus::zpz<367>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<222>, ZPZV<321>, ZPZV<324>, ZPZV<6»; }; // NOLINT
04452
           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<1>, ZPZV<361»; }; // NOLINT template<> struct ConwayPolynomial<367, 8> { using ZPZ = aerobus::zpz<367>; using type =
04453
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<335>, ZPZV<282>, ZPZV<50>, ZPZV<6»; };
           template<> struct ConwayPolynomial<367, 9> { using ZPZ = aerobus::zpz<367>; using type =
04454
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<213>, ZPZV<268>, ZPZV<361»;
       }; // NOLINT
04455
            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 =
      POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2»; }; // NOLINT
04457
           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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304>, ZPZV<2»; }; // NOLINT
04458
```

```
04459
           template<> struct ConwayPolynomial<373, 5> { using ZPZ = aerobus::zpz<373>; using type =
      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<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<83>, ZPZV<108>, ZPZV<2x; }; // NOLINT template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type
04461
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<371»; };
                                                                                                     // NOLINT
           template<> struct ConwayPolynomial<373, 8> { using ZPZ = aerobus::zpz<373>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<203>, ZPZV<219>, ZPZV<66>, ZPZV<2»; };
      template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<238>, ZPZV<370>, ZPZV<371»;
04463
       }; // NOLINT
           template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
04464
       POLYV<ZPZV<1>, ZPZV<377»; }; // NOLINT
04465
           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 =
04466
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377»; };
                                                            // NOLINT
           template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
04467
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<379, 5> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<377»; }; // NOLINT
           template<> struct ConwayPolynomial<379, 6> { using ZPZ = aerobus::zpz<379>; using type =
04469
      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
04470
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<377%; }; // NOLINT
04471
           template<> struct ConwayPolynomial<379, 8> { using ZPZ = aerobus::zpz<379>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<210>, ZPZV<194>, ZPZV<173>, ZPZV<2»; }; //
       NOLINT
04472
           template<> struct ConwayPolynomial<379, 9> { using ZPZ = aerobus::zpz<379>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<362>, ZPZV<369>, ZPZV<377»;
       }; // NOLINT
04473
           template<> struct ConwayPolynomial<383, 1> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<378»; }; // NOLINT
04474
           template<> struct ConwayPolynomial<383, 2> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
04475
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
04476
           template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<7>, ZPZY<309>, ZPZY<5»; }; // NOLINT template<> struct ConwayPolynomial<383, 5> { using ZPZ = aerobus::zpz<383>; using type =
04477
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
04478
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<8>, ZPZV<8158>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 7> { using ZPZ = aerobus::zpz<383>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<378»; };
04480
          template<> struct ConwayPolynomial<383, 8> { using ZPZ = aerobus::zpz<383>; using type =
       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 =
04481
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<137>, ZPZV<16>, ZPZV<378»;
       }; // NOLINT
04482
           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 =
04483
      POLYV<ZPZV<1>, ZPZV<379>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387»; }; // NOLINT template<>> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
04485
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<266>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
04486
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<387»; }; // NOLINT
04487
           template<> struct ConwayPolynomial<389, 6> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<339>, ZPZV<255>, ZPZV<2»; }; // NOLINT
04488
           template<> struct ConwayPolynomial<389, 7> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<387»; }; // NOLINT template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::zpz<389>; using type =
04489
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<351>, ZPZV<19>, ZPZV<290>, ZPZV<2»; }; //
       NOLINT
04490
           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
04491
           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 =
04492
      POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<397, 3> { using ZPZ = aerobus::zpz<397>; using type =
04493
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<392»; }; // NOLINT template<> struct ConwayPolynomial<397, 4> { using ZPZ = aerobus::zpz<397>; using type =
04494
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
04495
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392»; }; // NOLINT
           template<> struct ConwayPolynomial<397, 6> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<274>, ZPZV<287>, ZPZV<5»; }; // NOLINT
04497
          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<0>, ZPZV<2>, ZPZV<2>, ZPZV<392»; ); // NOLINT template<> struct ConwayPolynomial<397, 8> { using ZPZ = aerobus::zpz<397>; using type =
04498
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<375>, ZPZV<255>, ZPZV<203>, ZPZV<5»; }; //
04499
           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<66>, ZPZV<166>, ZPZV<252>, ZPZV<392»;
       }; // NOLINT
            template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
04500
       POLYV<ZPZV<1>, ZPZV<398»; }; // NOLINT
04501
            template<> struct ConwayPolynomial<401, 2> { using ZPZ = aerobus::zpz<401>; using type =
       POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3»; }; // NOLINT
04502
           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 =
04503
       POLYV<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<372>, ZPZV<3%; }; // NOLINT template<> struct ConwayPolynomial<401, 5> { using ZPZ = aerobus::zpz<401>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<398»; }; // NOLINT
04505
           template<> struct ConwayPolynomial<401, 6> { using ZPZ = aerobus::zpz<401>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<115>, ZPZV<81>, ZPZV<51>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<401, 7> { using ZPZ = aerobus::zpz<401>; using type
04506
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<398»; }; // NOLINT
            template<> struct ConwayPolynomial<401, 8> { using ZPZ = aerobus::zpz<401>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<380>, ZPZV<113>, ZPZV<164>, ZPZV<3%; }; //
       NOLINT
04508
           template<> struct ConwayPolynomial<401, 9> { using ZPZ = aerobus::zpz<401>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<158>, ZPZV<398»;
       }; // NOLINT
04509
            template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
       POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
04510
            template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
       POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT
           template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
04511
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388»; }; // NOLINT
04512
            template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<407>, ZPZV<21»; }; // NOLINT
04513
           template<> struct ConwayPolynomial<409, 5> { using ZPZ = aerobus::zpz<409>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<388»; }; // NOLINT template<> struct ConwayPolynomial<409, 6> { using ZPZ = aerobus::zpz<409>; using type =
04514
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<372>, ZPZV<364>, ZPZV<21»; }; // NOLINT template<> struct ConwayPolynomial<409, 7> { using ZPZ = aerobus::zpz<409>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<388»; };
           template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
04516
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<256>, ZPZV<69>, ZPZV<396>, ZPZV<31»; }; //
       NOLINT
           template<> struct ConwayPolynomial<409, 9> { using ZPZ = aerobus::zpz<409>; using type =
04517
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<318>, ZPZV<318>, ZPZV<318>,
       }; // NOLINT
04518
            template<> struct ConwayPolynomial<419, 1> { using ZPZ = aerobus::zpz<419>; using type =
       POLYV<ZPZV<1>, ZPZV<417»; }; // NOLINT
           template<> struct ConwayPolynomial<419, 2> { using ZPZ = aerobus::zpz<419>; using type =
04519
       POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
04520
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type =
       04522
           template<> struct ConwayPolynomial<419, 5> { using ZPZ = aerobus::zpz<419>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; // NOLINT
            template<> struct ConwayPolynomial<419, 6> { using ZPZ = aerobus::zpz<419>; using type =
04523
       POLYV<2PZV<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
04524
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 8> { using ZPZ = aerobus::zpz<419>; using type =
04525
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<234>, ZPZV<388>, ZPZV<151>, ZPZV<2»; }; //
       NOLINT
04526
            template<> struct ConwayPolynomial<419, 9> { using ZPZ = aerobus::zpz<419>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3, ZPZV<3, ZPZV<386>, ZPZV<417»;
       }; // NOLINT
04527
           template<> struct ConwayPolynomial<421, 1> { using ZPZ = aerobus::zpz<421>; using type =
       POLYV<ZPZV<1>, ZPZV<419»; }; // NOLINT
           template<> struct ConwayPolynomial<421, 2> { using ZPZ = aerobus::zpz<421>; using type =
04528
       POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial4421, 3> { using ZPZ = aerobus::zpz<421>; using type =
04529
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419»; }; // NOLINT template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
04530
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<421, 5> { using ZPZ = aerobus::zpz<421>; using type =
04531
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<419»; }; // NOLINT 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<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>; using type = POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<389>, ZPZV<32>, ZPZV<77>, ZPZV<2»; };
04533
                                                                                                           // NOLINT
04534
       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»;
       }; // NOLINT
            template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
04536
       POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
```

```
04537
           template<> struct ConwayPolynomial<431, 2> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7»; }; // NOLINT
04538
          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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7»; }; // NOLINT
04539
           template<> struct ConwayPolynomial 431, 5> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<431, 6> { using ZPZ = aerobus::zpz<431>; using type =
04541
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<161>, ZPZV<202>, ZPZV<182>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<431, 7> { using ZPZ = aerobus::zpz<431>; using type =
04542
      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 =
04543
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<243>, ZPZV<286>, ZPZV<115>, ZPZV<7»; }; //
      NOLINT
           template<> struct ConwayPolynomial<431, 9> { using ZPZ = aerobus::zpz<431>; using type =
04544
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<71>, ZPZV<329>, ZPZV<424%;
      }; // NOLINT
           template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<428»; }; // NOLINT
           template<> struct ConwayPolynomial<433, 2> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5»; }; // NOLINT
04547
           \texttt{template<>} \texttt{struct ConwayPolynomial<433, 3> \{ \texttt{using ZPZ = aerobus::zpz<433>; using type = 1.5 \}}
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428»; }; // NOLINT template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
04548
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<402>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<433, 5> { using ZPZ = aerobus::zpz<433>; using type =
04549
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<428»; }; // NOLINT
04550
           template<> struct ConwayPolynomial<433, 6> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<244>, ZPZV<353>, ZPZV<360>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<433, 7> { using ZPZ = aerobus::zpz<433>; using type
04551
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<428»; };
           template<> struct ConwayPolynomial<433, 8> { using ZPZ = aerobus::zpz<433>; using type
04552
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347>, ZPZV<32>, ZPZV<39>, ZPZV<5»; };
      NOLINT
           template<> struct ConwayPolynomial<433, 9> { using ZPZ = aerobus::zpz<433>; using type =
04553
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<23>, ZPZV<232>, ZPZV<45>, ZPZV<428»;
04554
           template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
04555
          template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; }; // NOLINT

template<> struct ConwayPolynomial</br>
439, 3> { using ZPZ = aerobus::zpz<439>; using type =
04556
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; }; // NOLINT
04558
          template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 6> { using ZPZ = aerobus::zpz<439>; using type =
04559
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<324>, ZPZV<190>, ZPZV<15»; }; // NOLINT
04560
           template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; //
04561
          template<> struct ConwayPolynomial<439, 8> { using ZPZ = aerobus::zpz<439>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<359>, ZPZV<296>, ZPZV<266>, ZPZV<268>, ZPZV<15»; }; //
      NOLINT
      template<> struct ConwayPolynomial<439, 9> { using ZPZ = aerobus::zpz<439>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<342>, ZPZV<342>, ZPZV<254>, ZPZV<424»;</pre>
04562
      }; // NOLINT
           template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
04563
      POLYV<ZPZV<1>, ZPZV<441»; }; // NOLINT
          template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
04564
      POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; }; // NOLINT
04565
           template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
04566
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2»; }; // NOLINT
04567
           template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44), ZPZV<441»; }; // NOLINT
           template<> struct ConwayPolynomial<443, 6> { using ZPZ = aerobus::zpz<443>; using type =
04568
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<298>, ZPZV<218>, ZPZV<41>, ZPZV<2»; };
           template<> struct ConwayPolynomial<443, 7> { using ZPZ = aerobus::zpz<443>; using type =
04569
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<64+1»; }; // NOLINT
04570
           template<> struct ConwayPolynomial<443, 8> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<217>, ZPZV<290>, ZPZV<2»: }; //
      NOLINT
           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<109>, ZPZV<441»;
      }; // NOLINT
04572
           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 =
04573
      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 =
04575
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<24>, ZPZV<24>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<449, 5> { using ZPZ = aerobus::zpz<449>; using type =
04576
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<446»; };
               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 =
04578
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<446»; }; // NOLINT template<> struct ConwayPolynomial<449, 8> { using ZPZ = aerobus::zpz<449>; using type =
04579
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361>, ZPZV<348>, ZPZV<124>, ZPZV<3»; }; //
04580
              template<> struct ConwayPolynomial<449, 9> { using ZPZ = aerobus::zpz<449>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<26>, ZPZV<26>, ZPZV<9>, ZPZV<446»; };
         // NOLINT
04581
               template<> struct ConwayPolynomial<457, 1> { using ZPZ = aerobus::zpz<457>; using type =
        POLYV<ZPZV<1>, ZPZV<444»; }; // NOLINT
               template<> struct ConwayPolynomial<457, 2> { using ZPZ = aerobus::zpz<457>; using type =
        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
04583
               template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
04584
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13»; }; // NOLINT
               template<> struct ConwayPolynomial<457, 5> { using ZPZ = aerobus::zpz<457>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<444»; }; // NOLINT
04586
               template<> struct ConwayPolynomial<457, 6> { using ZPZ = aerobus::zpz<457>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<205>, ZPZV<389>, ZPZV<266>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<457, 7> { using ZPZ = aerobus::zpz<457>; using type
04587
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<144, ZPZV<444*; }; // NOLINT
               template<> struct ConwayPolynomial<457, 8> { using ZPZ = aerobus::zpz<457>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<365>, ZPZV<296>, ZPZV<412>, ZPZV<13»; }; //
         NOLINT
        template<> struct ConwayPolynomial<457, 9> { using ZPZ = aerobus::zpz<457>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<354>, ZPZV<354>, ZPZV<3444»;
04589
         }; // NOLINT
04590
               template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
         POLYV<ZPZV<1>, ZPZV<459»; }; // NOLINT
04591
              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 =
04592
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZV<459»; }; // NOLINT template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
04594
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<459»; }; // NOLINT
04595
               template<> struct ConwayPolynomial<461, 6> { using ZPZ = aerobus::zpz<461>; using type =
        POLYYCZPZVC1>, ZPZVC0>, ZPZVC1>, ZPZVC439>, ZPZVC329>, 
04596
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<459»; }; // NOLINT
04597
               template<> struct ConwayPolynomial<461, 8> { using ZPZ = aerobus::zpz<461>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<449>, ZPZV<321>, ZPZV<32; }; //
         NOLINT
              template<> struct ConwayPolynomial<461, 9> { using ZPZ = aerobus::zpz<461>; using type =
04598
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<210>, ZPZV<276>, ZPZV<459»;
        }; // NOLINT template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
04599
        POLYV<ZPZV<1>, ZPZV<460»; }; // NOLINT template<> struct ConwayPolynomial<463, 2> { using ZPZ = aerobus::zpz<463>; using type =
04600
        POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
04601
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<460»; }; // NOLINT
              template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
04602
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
04603
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<4>, ZPZY<40, ZPZY<40, ZPZY<40, ZPZY<460x; }; // NOLINT template<> struct ConwayPolynomial<463, 6> { using ZPZ = aerobus::zpz<463>; using type =
04604
        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
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<13>, ZPZV<460»; };
04606
              template<> struct ConwayPolynomial<463, 8> { using ZPZ = aerobus::zpz<463>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<234>, ZPZV<414>, ZPZV<396>, ZPZV<3»; }; //
         NOLINT
              template<> struct ConwayPolynomial<463, 9> { using ZPZ = aerobus::zpz<463>; using type =
04607
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<433>, ZPZV<4227>, ZPZV<460»;
         }; // NOLINT
04608
              template<> struct ConwayPolynomial<467, 1> { using ZPZ = aerobus::zpz<467>; using type =
        POLYV<ZPZV<1>, ZPZV<465»; }; // NOLINT
               template<> struct ConwayPolynomial<467, 2> { using ZPZ = aerobus::zpz<467>; using type =
04609
        POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZV<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
04611
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<467, 5> { using ZPZ = aerobus::zpz<467>; using type =
04612
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
04613
               template<> struct ConwayPolynomial<467, 6> { using ZPZ = aerobus::zpz<467>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<123>, ZPZV<62>, ZPZV<237>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type =
04614
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 8> { using ZPZ = aerobus::zpz<467>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<318>, ZPZV<413>, ZPZV<289>, ZPZV<2»; }; //
04615
```

```
NOLINT
           template<> struct ConwayPolynomial<467, 9> { using ZPZ = aerobus::zpz<467>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<397>, ZPZV<447>, ZPZV<447>, ZPZV<447>,
04616
           }; // NOLINT
04617
                   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 =
           POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; }; // NOLINT
04619
                  template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466»; }; // NOLINT template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
04620
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<479, 5> { using ZPZ = aerobus::zpz<479>; using type =
04621
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<466»; }; // NOLINT
04622
                  template<> struct ConwayPolynomial<479, 6> { using ZPZ = aerobus::zpz<479>; using type =
           POLYV<ZPZV<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
04623
          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
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<247>, ZPZV<440>, ZPZV<17>, ZPZV<13»; }; //
04625
                  template<> struct ConwayPolynomial<479, 9> { using ZPZ = aerobus::zpz<479>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<185>, ZPZV<466»; };
           // NOLINT
04626
                   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 =
04627
           POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; }; // NOLINT
04628
                  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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3»; }; // NOLINT
04629
                  template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
04630
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<484»; }; // NOLINT
04631
                  template<> struct ConwayPolynomial<487, 6> { using ZPZ = aerobus::zpz<487>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<427>, ZPZV<185>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<487, 7> { using ZPZ = aerobus::zpz<487>; using type
04632
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<484»; }; //
                  template<> struct ConwayPolynomial<487, 8> { using ZPZ = aerobus::zpz<487>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<249>, ZPZV<137>, ZPZV<3»; }; //
           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<484*);</pre>
04634
           }; // NOLINT
                   template<> struct ConwayPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
           POLYV<ZPZV<1>, ZPZV<489»; }; // NOLINT
04636
                  template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
          POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; }; // NOLINT
04637
                  template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type =
04638
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360>, ZPZV<2»; }; // NOLINT
04639
                  template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 6> { using ZPZ = aerobus::zpz<491>; using type =
04640
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<369>, ZPZV<42>, ZPZV<125>, ZPZV<22»; }; // NOLINT template<> struct ConwayPolynomial<491, 7> { using ZPZ = aerobus::zpz<491>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; };
                  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*; }; //
           NOLINT
           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<453>, ZPZV<453>, ZPZV<489»;
04643
                  template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
04644
           POLYV<ZPZV<1>, ZPZV<492»; }; // NOLINT
04645
                   template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
           POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; }; // NOLINT
                  template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
04646
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
04647
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; }; // NOLINT
                  template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
04648
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<492»; }; // NOLINT
                  template<> struct ConwayPolynomial<499, 6> { using ZPZ = aerobus::zpz<499>; using type =
04649
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<407>, ZPZV<191>, ZPZV<78>, ZPZV<7»; }; // NOLINI
                  template<> struct ConwayPolynomial<499, 7> { using ZPZ = aerobus::zpz<499>; using type
04650
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 8> { using ZPZ = aerobus::zpz<499>; using type =
04651
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<288>, ZPZV<309>, ZPZV<200>, ZPZV<7»; }; //
           NOLINT
04652
                  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<5>, ZPZV<491>, ZPZV<222>, ZPZV<492»;
           }; // NOLINT
04653
                  \texttt{template<> struct ConwayPolynomial<503, 1> \{ using ZPZ = aerobus:: zpz<503>; using type = 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 +
          POLYV<ZPZV<1>, ZPZV<498»; }; // NOLINT
                  template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
04654
```

```
POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; };
            template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<498»; }; // NOLINT template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type =
04656
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<325>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<503, 5> { using ZPZ = aerobus::zpz<503>; using type =
04657
       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 =
04658
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<380>, ZPZV<292>, ZPZV<255>, ZPZV<55»; }; // NOLINT template<> struct ConwayPolynomial<503, 7> { using ZPZ = aerobus::zpz<503>; using type =
04659
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<498»; }; // NOLINT template<> struct ConwayPolynomial<503, 8> { using ZPZ = aerobus::zpz<503>; using type =
04660
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<441>, ZPZV<203>, ZPZV<316>, ZPZV<5»; }; //
04661
           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<3>, ZPZV<3>, ZPZV<158>, ZPZV<337>, ZPZV<498»;
       }; // NOLINT
04662
            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 =
       POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2»; }; // NOLINT
04664
           template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2»; }; // NOLINT
04665
            template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
04667
           template<> struct ConwayPolynomial<509, 6> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<350>, ZPZV<232>, ZPZV<41>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<509, 7> { using ZPZ = aerobus::zpz<509>; using type =
04668
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>); // NOLINT template<> struct ConwayPolynomial<509, 8> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<420>, ZPZV<473>, ZPZV<382>, ZPZV<382>, //
      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<314>, ZPZV<28>, ZPZV<507»;</pre>
04670
       }; // NOLINT
            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 =
      POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
04673
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518»; }; // NOLINT template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
04674
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<509>, ZPZV<3»; }; // NOLINT
04675
           template<> struct ConwayPolynomial<521, 5> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<518»; }; // NOLINT
04676
           template<> struct ConwayPolynomial<521, 6> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<315>, ZPZV<153>, ZPZV<280>, ZPZV<3w; }; // NOLINT template<> struct ConwayPolynomial<521, 7> { using ZPZ = aerobus::zpz<521>; using type =
04677
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<518»; };
           template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<462>, ZPZV<407>, ZPZV<312>, ZPZV<3»; }; //
04679
           template<> struct ConwayPolynomial<521, 9> { using ZPZ = aerobus::zpz<521>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<181>, ZPZV<483>, ZPZV<518»;
       }; // NOLINT
    template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
04680
       POLYV<ZPZV<1>, ZPZV<521»; }; // NOLINT
            template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
04681
       POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
04682
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
04684
       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
04685
            template<> struct ConwayPolynomial<523, 7> { using ZPZ = aerobus::zpz<523>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<521»; };
           template<> struct ConwayPolynomial<523, 8> { using ZPZ = aerobus::zpz<523>; using type =
04687
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<518>, ZPZV<184>, ZPZV<380>, ZPZV<2»; }; //
       NOLINT
           template<> struct ConwayPolynomial<523, 9> { using ZPZ = aerobus::zpz<523>; using type =
04688
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<342>, ZPZV<145>, ZPZV<521»;
       }; // NOLINT
04689
           template<> struct ConwayPolynomial<541, 1> { using ZPZ = aerobus::zpz<541>; using type =
       POLYV<ZPZV<1>, ZPZV<539»; }; // NOLINT
           template<> struct ConwayPolynomial<541, 2> { using ZPZ = aerobus::zpz<541>; using type =
04690
       POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539»; }; // NOLINT template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
04692
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
04693
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<539»; }; // NOLINT
```

```
04694
                template<> struct ConwayPolynomial<541, 6> { using ZPZ = aerobus::zpz<541>; using type =
         POLYY<ZPZY<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 =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<539»; }; // NOLINT template<> struct ConwayPolynomial<541, 8> { using ZPZ = aerobus::zpz<541>; using type =
04696
         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
04697
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<340>, ZPZV<318>, ZPZV<539»;
         }; // NOLINT
04698
               template<> struct ConwayPolynomial<547, 1> { using ZPZ = aerobus::zpz<547>; using type =
         POLYV<ZPZV<1>, ZPZV<545»; }; // NOLINT
               template<> struct ConwayPolynomial<547, 2> { using ZPZ = aerobus::zpz<547>; using type =
04699
         POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; }; // NOLINT
04700
               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
template<> struct ConwayPolynomial<547, 5> { using ZPZ = aerobus::zpz<547>; using type =
04701
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<545»; }; // NOLINT
                template<> struct ConwayPolynomial<547, 6> { using ZPZ = aerobus::zpz<547>; using type =
04703
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<2»; }; // NOLINT
04704
               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<5, Z
04705
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2»; }; //
         template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<238>, ZPZV<263>, ZPZV<245»;
04706
         }; // NOLINT
               template<> struct ConwayPolynomial<557, 1> { using ZPZ = aerobus::zpz<557>; using type =
04707
         POLYV<ZPZV<1>, ZPZV<555»; }; // NOLINT
               template<> struct ConwayPolynomial<557, 2> { using ZPZ = aerobus::zpz<557>; using type =
04708
         POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; }; // NOLINT
04709
                template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<555»; }; // NOLINT
template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; }; // NOLINT
04710
04711
                template<> struct ConwayPolynomial<557, 5> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<555»; }; // NOLINT
         template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<202>, ZPZV<192>, ZPZV<253>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<557, 7> { using ZPZ = aerobus::zpz<557>; using type =
04712
04713
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<555»; };
               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<12»; }; //
         NOLINT
         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<436>, ZPZV<434>, ZPZV<555»;</pre>
04715
         }; // NOLINT
04716
                template<> struct ConwayPolynomial<563, 1> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<561»; }; // NOLINT
04717
                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 =
04718
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; }; // NOLINT template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<20>, ZPZV<399>, ZPZV<2»; }; // NOLINT
                template<> struct ConwayPolynomial<563, 5> { using ZPZ = aerobus::zpz<563>; using type =
04720
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<561»; }; // NOLINT
               template<> struct ConwayPolynomial<563, 6> { using ZPZ = aerobus::zpz<563>; using type =
04721
         POLYVCZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<122>, ZPZV<246>, ZPZV<24); ; // NOLINT template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz<563>; using type
04722
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<561»; }; //
04723
               template<> struct ConwayPolynomial<563, 8> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<176>, ZPZV<509>, ZPZV<2»; }; //
         NOLINT
04724
               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<19>, ZPZV<561»; };
          // NOLINT
                template<> struct ConwayPolynomial<569, 1> { using ZPZ = aerobus::zpz<569>; using type =
04725
         POLYV<ZPZV<1>, ZPZV<566»; }; // NOLINT
               template<> struct ConwayPolynomial<569, 2> { using ZPZ = aerobus::zpz<569>; using type =
04726
         POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
04727
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT
               template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
04728
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<569, 5> { using ZPZ = aerobus::zpz<569>; using type =
04729
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT
               template<> struct ConwayPolynomial<569, 6> { using ZPZ = aerobus::zpz<569>; using type =
04730
         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 =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<566»; };
04732
               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»; }; //
         NOLTNT
```

```
template<> struct ConwayPolynomial<569, 9> { using ZPZ = aerobus::zpz<569>; using type :
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<478>, ZPZV<566>, ZPZV<566»;
       }; // NOLINT
04734
           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 =
04735
       POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
04736
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type =
04737
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3»; }; // NOLINT

template<> struct ConwayPolynomial<571, 5> { using ZPZ = aerobus::zpz<571>; using type =
04738
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 6> { using ZPZ = aerobus::zpz<571>; using type =
04739
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<221>, ZPZV<295>, ZPZV<33>, ZPZV<3»; }; // NOLINT
04740
           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<7>, ZPZV<568»; }; // NOLINT
           template<> struct ConwayPolynomial<571, 8> { using ZPZ = aerobus::zpz<571>; using type =
04741
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<363>, ZPZV<119>, ZPZV<371>, ZPZV<3»; }; //
       template<> struct ConwayPolynomial<571, 9> { using ZPZ = aerobus::zpz<571>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<545>, ZPZV<545>, ZPZV<179>, ZPZV<568»;
04742
       }; // NOLINT
            template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
04743
       POLYV<ZPZV<1>, ZPZV<572»; }; // NOLINT
            template<> struct ConwayPolynomial<577, 2> { using ZPZ = aerobus::zpz<577>; using type =
       POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
04745
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572»; }; // NOLINT
template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<577, 5> { using ZPZ = aerobus::zpz<577>; using type =
04746
04747
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<572»; }; // NOLINT
04748
           template<> struct ConwayPolynomial<577, 6> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<45O>, ZPZV<25>, ZPZV<283>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<577, 7> { using ZPZ = aerobus::zpz<577>; using type =
04749
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<572»; }; // NOLINT template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<5»; };
04751
           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<0>, ZPZV<17>, ZPZV<576>, ZPZV<449>, ZPZV<572»;
       }; // NOLINT
04752
            template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
       POLYV<ZPZV<1>, ZPZV<585»; }; // NOLINT
04753
            template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
       POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2»; }; // NOLINT
04754
           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 =
04755
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<444>, ZPZV<2»; };
                                                                             // NOLINT
            template<> struct ConwayPolynomial<587, 5> { using ZPZ = aerobus::zpz<587>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<585»; }; // NOLINT
      template<> struct ConwayPolynomial<587, 6> { using ZPZ = aerobus::zpz<587>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<204>, ZPZV<121>, ZPZV<226>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type =
04757
04758
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<585»; }; //
           template<> struct ConwayPolynomial<587, 8> { using ZPZ = aerobus::zpz<587>; using type =
04759
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<492>, ZPZV<444>, ZPZV<91>, ZPZV<2»; };
       NOLINT
04760
           template<> struct ConwayPolynomial<587, 9> { using ZPZ = aerobus::zpz<587>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<333>, ZPZV<55>, ZPZV<585»;
       }; // NOLINT
            template<> struct ConwayPolynomial<593, 1> { using ZPZ = aerobus::zpz<593>; using type =
       POLYV<ZPZV<1>, ZPZV<590»; }; // NOLINT
04762
           template<> struct ConwayPolynomial<593, 2> { using ZPZ = aerobus::zpz<593>; using type =
       POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<593, 3> { using ZPZ = aerobus::zpz<593>; using type =
04763
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<590»; }; // NOLINT
            template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
04764
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<593, 5> { using ZPZ = aerobus::zpz<593>; using type =
04765
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<590»; }; // NOLINT template<> struct ConwayPolynomial<593, 6> { using ZPZ = aerobus::zpz<593>; using type =
04766
      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 =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<590»; };
04768
           template<> struct ConwayPolynomial<593, 8> { using ZPZ = aerobus::zpz<593>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<350>, ZPZV<291>, ZPZV<495>, ZPZV<49s, }; //
       NOLINT
04769
            template<> struct ConwayPolynomial<593, 9> { using ZPZ = aerobus::zpz<593>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223>, ZPZV<523>, ZPZV<590»;
       }; // NOLINT
04770
            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
```

```
template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
        POLYY<ZPZY<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
              template<> struct ConwayPolynomial<599, 5> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<592»; }; // NOLINT
              template<> struct ConwayPolynomial<599, 6> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<2PZV<1>, 2PZV<0>, 2PZV<1>, 2PZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7»; }; // NOLINI
04776
             template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592»; }; // NOLINT
04777
              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»; }; //
               template<> struct ConwayPolynomial<599, 9> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<314>, ZPZV<98>, ZPZV<592»;
        }; // NOLINT
04779
              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 =
        POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
               template<> struct ConwayPolynomial<601, 3> { using ZPZ = aerobus::zpz<601>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<594»; }; // NOLINT template<> struct ConwayPolynomial<601, 4> { using ZPZ = aerobus::zpz<601>; using type =
04782
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<601, 5> { using ZPZ = aerobus::zpz<601>; using type =
04783
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<594»; }; // NOLINT
04784
              template<> struct ConwayPolynomial<601, 6> { using ZPZ = aerobus::zpz<601>; using type =
        04785
               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 =
04786
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<550>, ZPZV<241>, ZPZV<490>, ZPZV<7»; }; //
             template<> struct ConwayPolynomial<601, 9> { using ZPZ = aerobus::zpz<601>; using type =
04787
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<487>, ZPZV<590>, ZPZV<594*;
        }; // NOLINT
        template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
POLYV<ZPZV<1>, ZPZV<604»; }; // NOLINT</pre>
04788
               template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
        POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; }; // NOLINT
04790
              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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3»; }; // NOLINT
04791
               template<> struct ConwayPolynomial<607, 5> { using ZPZ = aerobus::zpz<607>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<604»; }; // NOLINT
04793
              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<3»; }; // NOLINT template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type =
04794
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<604»; }; // NOLINT
              template<> struct ConwayPolynomial<607, 8> { using ZPZ = aerobus::zpz<607>; using type
04795
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<468>, ZPZV<35>, ZPZV<449>, ZPZV<33»; };
        NOLINT
        template<> struct ConwayPolynomial<607, 9> { using ZPZ = aerobus::zpz<607>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<444>, ZPZV<444>, ZPZV<129>, ZPZV<604*;</pre>
04796
        }; // NOLINT
04797
               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 =
04798
        POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
04799
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<61>; }; // NOLINT template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
04800
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
04801
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<611»; }; // NOLINT
04802
              template<> struct ConwayPolynomial<613, 6> { using ZPZ = aerobus::zpz<613>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<609>, ZPZV<601>, ZPZV<601>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type
04803
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<61»; };
              template<> struct ConwayPolynomial<613, 8> { using ZPZ = aerobus::zpz<613>; using type =
04804
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<489>, ZPZV<57>, ZPZV<539>, ZPZV<2»; };
        NOLINT
04805
              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<513>, ZPZV<536>, ZPZV<611»;
        }; // NOLINT
04806
               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 =
04807
        POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
04808
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<614»; }; // NOLINT
        template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3»; }; // NOLINT
04810
             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 =
04811
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<318>, ZPZV<595>, ZPZV<310>, ZPZV<3»; };
           template<> struct ConwayPolynomial<617, 7> { using ZPZ = aerobus::zpz<617>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<614»; }; // NOLINT
           template<> struct ConwayPolynomial<617, 8> { using ZPZ = aerobus::zpz<617>; using type =
04813
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<519>, ZPZV<501>, ZPZV<155>, ZPZV<3»; }; //
       NOLTNT
       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»;
04814
       }; // NOLINT
04815
           template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
       POLYV<ZPZV<1>, ZPZV<617»; }; // NOLINT
           template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
04816
       POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
04817
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617»; }; // NOLINT template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
04818
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<619, 5> { using ZPZ = aerobus::zpz<619>; using type =
04819
       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 template<> struct ConwayPolynomial<619, 7> { using ZPZ = aerobus::zpz<619>; using type =
04821
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6
| NoLINT template<> struct ConwayPolynomial<619, 8> { using ZPZ = aerobus::zpz<619>; using type =
04822
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<416>, ZPZV<383>, ZPZV<225>, ZPZV<2»; }; //
           template<> struct ConwayPolynomial<619, 9> { using ZPZ = aerobus::zpz<619>; using type
04823
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<510>, ZPZV<617»;
       }; // NOLINT
04824
           template<> struct ConwayPolynomial<631, 1> { using ZPZ = aerobus::zpz<631>; using type =
       POLYV<ZPZV<1>, ZPZV<628»; }; // NOLINT
04825
            template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
       POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3»; }; // NOLINT
04826
           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 =
04827
       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<3%; }; // NOLINT template<> struct ConwayPolynomial<631, 5> { using ZPZ = aerobus::zpz<631>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
           template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<516>, ZPZV<541>, ZPZV<106>, ZPZV<3»; }; // NOLINT
04830
           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 =
04831
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<3»; }; //
       NOLINT
04832
           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
04833
            template<> struct ConwavPolynomial<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 =
       POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
04835
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<638»; }; // NOLINT template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<3»; }; // NOLINT
04836
           template<> struct ConwayPolynomial<641, 5> { using ZPZ = aerobus::zpz<641>; using type =
04837
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<638»; }; // NOLINT
04838
            template<> struct ConwayPolynomial<641, 6> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<557>, ZPZV<294>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type
04839
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<638»; }; // NOLINT
           template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>;
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<33»; }; //
       NOLINT
04841
       template<> struct ConwayPolynomial<641, 9> { using ZPZ = aerobus::zpz<641>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<141>, ZPZV<638»;</pre>
       }; // NOLINT
04842
            template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
       POLYV<ZPZV<1>, ZPZV<632»; }; // NOLINT
04843
           template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
       POLYV<ZPZV<1>, ZPZV<641>, ZPZV<611»; }; // NOLINT template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
04844
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<632»; }; // 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
04846
            template<> struct ConwayPolynomial<643, 5> { using ZPZ = aerobus::zpz<643>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<632»; }; // NOLINT
           template<> struct ConwayPolynomial<643, 6> { using ZPZ = aerobus::zpz<643>; using type =
04847
       POLYY<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
                                                          7> { using ZPZ = aerobus::zpz<643>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<632»; }; // NOLI template<> struct ConwayPolynomial<643, 8> { using ZPZ = aerobus::zpz<643>; using type
04849
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<531>, ZPZV<573>, ZPZV<569>, ZPZV<11»; }; //
       NOLINT
04850
           template<> struct ConwayPolynomial<643, 9> { using ZPZ = aerobus::zpz<643>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<591>, ZPZV<475>, ZPZV<632*;
04851
              template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<642»; }; // NOLINT
04852
              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 =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<642»; }; // NOLINT
             template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
04854
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<647, 5> { using ZPZ = aerobus::zpz<647>; using type =
04855
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<642»; }; // NOLINT
              template<> struct ConwayPolynomial<647, 6> { using ZPZ = aerobus::zpz<647>; using type =
04856
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<308>, ZPZV<385>, ZPZV<642>, ZPZV<6*; }; // NOLIN
04857
             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<0>, ZPZV<1>, ZPZV<642»; }; // NOLIN template<> struct ConwayPolynomial<647, 8> { using ZPZ = aerobus::zpz<647>; using type =
04858
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<603>, ZPZV<259>, ZPZV<271>, ZPZV<5»; }; //
04859
              template<> struct ConwayPolynomial<647, 9> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<123>, ZPZV<561>, ZPZV<562»;
        }; // NOLINT
04860
              template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<651»; }; // NOLINT
04861
              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 =
04862
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<651»; }; // NOLINT template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type =
04863
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2»; }; // NOLINT

template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
04864
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651»; }; // NOLINT
              template<> struct ConwayPolynomial<653, 6> { using ZPZ = aerobus::zpz<653>; using type =
04865
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<45>, ZPZV<220>, ZPZV<242>, ZPZV<242>, ZPZV<2»; }; // NOLINT
04866
              template<> struct ConwayPolynomial<653, 7> { 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<15>, ZPZV<651»; };
              template<> struct ConwayPolynomial<653, 8> { using ZPZ = aerobus::zpz<653>; using type =
04867
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<385>, ZPZV<18>, ZPZV<296>, ZPZV<2»; };
        NOLINT
              template<> struct ConwayPolynomial<653, 9> { using ZPZ = aerobus::zpz<653>; using type =
04868
        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
        }; // NOLINT
              template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
04869
        POLYV<ZPZV<1>, ZPZV<657»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2»; }; // NOLINT
04871
             template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<657»; }; // NOLINT template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
04872
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
04873
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<657»; }; // NOLINT
04874
              template<> struct ConwayPolynomial<659, 6> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<371>, ZPZV<105>, ZPZV<223>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<659, 7> { using ZPZ = aerobus::zpz<659>; using type =
04875
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<657»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 8> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<358>, ZPZV<246>, ZPZV<90>, ZPZV<2»; };
04877
              template<> struct ConwayPolynomial<659, 9> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<657»;
        }; // NOLINT
04878
              template<> struct ConwayPolynomial<661, 1> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<659»; }; // NOLINT
04879
             template<> struct ConwayPolynomial<661, 2> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2»; }; // NOLINT
04880
              template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<659»; }; // NOLINT template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type =
04881
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
04882
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<659»; }; // NOLINT
04883
              template<> struct ConwayPolynomial<661, 6> { using ZPZ = aerobus::zpz<661>; using type =
        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
04884
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<659»; }; // NOLINT
             template<> struct ConwayPolynomial<661, 8> { using ZPZ = aerobus::zpz<661>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<612>, ZPZV<285>, ZPZV<72>, ZPZV<22»; };
        NOLINT
04886
        template<> struct ConwayPolynomial<661, 9> { using ZPZ = aerobus::zpz<661>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<389>, ZPZV<389>, ZPZV<220>, ZPZV<659»;</pre>
        }; // NOLINT
    template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
        POLYV<ZPZV<1>, ZPZV<668»; }; // NOLINT
             template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
04888
       POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5»; }; // NOLINT
             template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
04889
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<668»; }; // NOLINT
      template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<673, 5> { using ZPZ = aerobus::zpz<673>; using type =
04891
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<668»; }; // NOLINT
      template<> struct ConwayPolynomial<673, 6> { using ZPZ = aerobus::zpz<673>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<524>, ZPZV<248>, ZPZV<35>, ZPZV<5»; }; // NOLINT
04892
           template<> struct ConwayPolynomial<673, 7> { using ZPZ = aerobus::zpz<673>; using type =
04893
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<66, ZPZV<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 8> { using ZPZ = aerobus::zpz<673>; using type =
04894
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<669>, ZPZV<587>, ZPZV<302>, ZPZV<5»; }; //
       NOLINT
           template<> struct ConwayPolynomial<673, 9> { using ZPZ = aerobus::zpz<673>; using type =
04895
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<347>, ZPZV<553>, ZPZV<668»;
       }; // NOLINT
04896
           template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<675»; }; // NOLINT
           template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
04897
       POLYV<ZPZV<1>, ZPZV<672>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<677, 3> { using ZPZ = aerobus::zpz<677>; using type =
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<675»; }; // NOLINT template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
04899
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2»; ); // NOLINT template<> struct ConwayPolynomial<677, 5> { using ZPZ = aerobus::zpz<677>; using type =
04900
      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<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446>, ZPZV<632>, ZPZV<50>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type =
04902
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<675»; };
           template<> struct ConwayPolynomial<677, 8> { using ZPZ = aerobus::zpz<677>; using type =
04903
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<619>, ZPZV<152>, ZPZV<2*; }; //
       NOLINT
           template<> struct ConwayPolynomial<677, 9> { using ZPZ = aerobus::zpz<677>; using type =
04904
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<504>, ZPZV<404>, ZPZV<675»;
       }; // NOLINT
04905
           template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<678»; }; // NOLINT
           template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
04907
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<678»; }; // NOLINT template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type =
04908
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<683, 5> { using ZPZ = aerobus::zpz<683>; using type =
04909
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<678»; }; // NOLINT
04910
           template<> struct ConwayPolynomial<683, 6> { using ZPZ = aerobus::zpz<683>; using type =
      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
04911
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<678»; }; // NOLINT
           template<> struct ConwayPolynomial<683, 8> { using ZPZ = aerobus::zpz<683>; using type =
04912
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<5»; };
04913
           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<0>, ZPZV<85>, ZPZV<444>, ZPZV<678»;
       }; // NOLINT
04914
           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 =
04915
       POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; }; // NOLINT
04916
           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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; }; // NOLINT
04917
           template<> struct ConwayPolynomial<691, 5> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; }; // NOLINT
04919
           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 template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type =
04920
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; }; // NOLINT
           template<> struct ConwayPolynomial<691, 8> { using ZPZ = aerobus::zpz<691>; using type
04921
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<356>, ZPZV<425>, ZPZV<321>, ZPZV<32), }; //
      04922
       }; // NOLINT
           template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<699»; }; // NOLINT
04924
           template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
04925
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; }; // NOLINT template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
04926
      POLYVCZPZV<1>, ZPZV<3>, ZPZV<12>, ZPZV<379>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
04927
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699»; }; // NOLINT
      template<> struct ConwayPolynomial<701, 6> { using ZPZ = aerobus::zpz<701>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<571>, ZPZV<327>, ZPZV<285>, ZPZV<2»; }; // NOLINT
04928
```

```
template<> struct ConwayPolynomial<701, 7> { using ZPZ = aerobus::zpz<701>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<699»; }; // NOLINT
04930
                 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<
          NOLINT
                 template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type =
04931
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<45>, ZPZV<459>, ZPZV<459>, ZPZV<699»;
          }; // NOLINT
04932
                 template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
          POLYV<ZPZV<1>, ZPZV<707»; }; // NOLINT
                  template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
04933
          POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
04934
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707»; }; // NOLINT
04935
                 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 =
04936
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10, ZPZV<10, ZPZV<10, ZPZV<707%; }; // NOLINT template<> struct ConwayPolynomial<709, 6> { using ZPZ = aerobus::zpz<709>; using type =
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<669>, ZPZV<514>, ZPZV<295>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<709, 7> { using ZPZ = aerobus::zpz<709>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<707»; }; // NOLINT
04939
                 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
                  template<> struct ConwayPolynomial<709, 9> { using ZPZ = aerobus::zpz<709>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<257>, ZPZV<2717>, ZPZV<707»;
          }; // NOLINT
04941
                  template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
          POLYV<ZPZV<1>, ZPZV<708»; }; // NOLINT
                 template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
04942
          POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; };
                                                                                  // NOLINT
                  template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type =
04943
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
                  template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
04944
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<602>, ZPZV<11»; }; // NOLINT
                  template<> struct ConwayPolynomial<719, 5> { using ZPZ = aerobus::zpz<719>; using type =
04945
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
04946
                  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 =
04947
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT template<> struct ConwayPolynomial<719, 8> { using ZPZ = aerobus::zpz<719>; using type =
04948
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<714>, ZPZV<362>, ZPZV<244>, ZPZV<211»; }; //
04949
                  template<> struct ConwayPolynomial<719, 9> { using ZPZ = aerobus::zpz<719>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<38>, ZPZV<288>, ZPZV<560>, ZPZV<708»;
          }; // NOLINT
04950
                  template<> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
          POLYV<ZPZV<1>, ZPZV<722»; }; // NOLINT
04951
                  template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
          POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; }; // NOLINT
04952
                  template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<722»; }; // NOLINT template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
04953
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<723>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<727, 5> { using ZPZ = aerobus::zpz<727>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<722»; }; // NOLINT
                  template<> struct ConwayPolynomial<727, 6> { using ZPZ = aerobus::zpz<727>; using type =
04955
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<86>, ZPZV<397>, ZPZV<672>, ZPZV<5»; }; // NOLINT
                 template<> struct ConwayPolynomial<727, 7> { using ZPZ = aerobus::zpz<727>; using type =
04956
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<12»; }; // NOLINT
                  template<> struct ConwayPolynomial<727, 8> { using ZPZ = aerobus::zpz<727>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<639>, ZPZV<671>, ZPZV<368>, ZPZV<36»; }; //
          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<573>, ZPZV<502>, ZPZV<722»;</pre>
04958
          }; // NOLINT
                  template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
04959
          POLYV<ZPZV<1>, ZPZV<727»; }; // NOLINT
                  template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
04960
          POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; }; // NOLINT
                  template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
04961
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727»; }; // NOLINT template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<6»; }; // NOLINT
04962
                 template<> struct ConwayPolynomial<733, 5> { using ZPZ = aerobus::zpz<733>; using type =
04963
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<727»; }; // NOLINT
04964
                  template<> struct ConwayPolynomial<733, 6> { using ZPZ = aerobus::zpz<733>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<549>, ZPZV<51>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<733, 7> { using ZPZ = aerobus::zpz<733>; using type = aerobus::zpz<7
04965
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<727»; };
                 template<> struct ConwayPolynomial<733, 8> { using ZPZ = aerobus::zpz<733>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<532>, ZPZV<610>, ZPZV<142>, ZPZV<6%; }; //
          template<> struct ConwayPolynomial<733, 9> { using ZPZ = aerobus::zpz<733>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<337>, ZPZV<6>, ZPZV<727»; };</pre>
04967
```

```
// NOLINT
04968
            template<> struct ConwayPolynomial<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 =
04969
       POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
04970
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; }; // NOLINT
04971
            template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
       template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
04972
      template<> struct ConwayPolynomial</39, 5> { using ZPZ = aerobus::zpz</39>; using type = PoLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<736»; }; // NOLINT template<> struct ConwayPolynomial</39, 6> { using ZPZ = aerobus::zpz</39>; using type = PoLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<422>, ZPZV<447>, ZPZV<625>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial</39, 7> { using ZPZ = aerobus::zpz<739>; using type =
04973
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<44>, ZPZV<736»; };
04975
           template<> struct ConwayPolynomial<739, 8> { using ZPZ = aerobus::zpz<739>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<401>, ZPZV<169>, ZPZV<25>, ZPZV<3»; }; //
       NOLINT
           template<> struct ConwayPolynomial<739, 9> { using ZPZ = aerobus::zpz<739>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<616>, ZPZV<81>, ZPZV<83»;
       }; // NOLINT
04977
            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 =
04978
       POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; }; // NOLINT
            template<> struct ConwayPolynomial</743, 3> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<738»; }; // NOLINT template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
04980
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<743, 5> { using ZPZ = aerobus::zpz<743>; using type =
04981
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT
04982
            template<> struct ConwayPolynomial<743, 6> { using ZPZ = aerobus::zpz<743>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<236>, ZPZV<471>, ZPZV<88>, ZPZV<5»; }; // NOLINI
04983
           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»; }; // NOLINT template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
04984
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>; }; //
04985
           template<> struct ConwayPolynomial<743, 9> { using ZPZ = aerobus::zpz<743>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<327>, ZPZV<676>, ZPZV<738»;
       }; // NOLINT
           template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
04986
       POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
           template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
04987
       POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT
04988
           template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; }; // NOLINT
template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<52>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<751, 5> { using ZPZ = aerobus::zpz<751>; using type =
04989
04990
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT
            template<> struct ConwayPolynomial<751, 6> { using ZPZ = aerobus::zpz<751>; using type =
04991
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<298>, ZPZV<633>, ZPZV<539>, ZPZV<3»; }; // NOLINT
04992
           template<> struct ConwayPolynomial<751, 7> { using ZPZ = aerobus::zpz<751>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT
           template<> struct ConwayPolynomial<751, 8> { using ZPZ = aerobus::zpz<751>; using type =
04993
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<741>, ZPZV<243>, ZPZV<672>, ZPZV<672>, ZPZV<3»; }; //
       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<70>, ZPZV<70>, ZPZV<48>;
       }; // NOLINT
            template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
04995
       POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
           template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
       POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
04997
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
04998
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<757, 5> { using ZPZ = aerobus::zpz<757>; using type =
04999
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<755»; }; // NOLINT
0.5000
           template<> struct ConwayPolynomial<757, 6> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<753>, ZPZV<745>, ZPZV<7*; // NOLINT template<> struct ConwayPolynomial<757, 7> { using ZPZ = aerobus::zpz<757>; using type =
05001
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, 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»; };
           template<> struct ConwayPolynomial<757, 9> { using ZPZ = aerobus::zpz<757>; using type =
05003
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<88>, ZPZV<688>, ZPZV<702>, ZPZV<755»;
       }; // NOLINT
            template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
       POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
05005
           template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<758>, ZPZV<66; }; // NOLINT template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755»; }; // NOLINT
05006
```

```
05007
            template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type =
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<6ss; }; // NOLINT template<> struct ConwayPolynomial<761, 5> { using ZPZ = aerobus::zpz<761>; using type =
05008
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT
      template<> struct ConwayPolynomial<761, 6> { using ZPZ = aerobus::zpz<761>; using type = POLYV<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 =
05009
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<755»; };
05011
          template<> struct ConwayPolynomial<761, 8> { using ZPZ = aerobus::zpz<761>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<603>, ZPZV<144>, ZPZV<540>, ZPZV<540>; }; //
       NOLINT
           template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type =
05012
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<317>, ZPZV<571>, ZPZV<755»;
       }; // NOLINT
05013
           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 =
05014
      POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 4> { using ZPZ = aerobus::zpz<769>; using type =
      05017
           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
05018
           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
05019
           template<> struct ConwayPolynomial<769, 7> { using ZPZ = aerobus::zpz<769>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<758»; }; // NOLINT template<> struct ConwayPolynomial<769, 8> { using ZPZ = aerobus::zpz<769>; using type =
05020
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<560>, ZPZV<574>, ZPZV<632>, ZPZV<11»; }; //
       NOLINT
05021
           template<> struct ConwayPolynomial<769, 9> { using ZPZ = aerobus::zpz<769>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<623>, ZPZV<751>, ZPZV<758»;
       }; // NOLINT
05022
           template<> struct ConwayPolynomial<773, 1> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<771»; }; // NOLINT
           template<> struct ConwayPolynomial<773, 2> { using ZPZ = aerobus::zpz<773>; using type =
05023
       POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»; }; // NOLINT
05024
           template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<771%; }; // NOLINT template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
05025
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<773, 5> { using ZPZ = aerobus::zpz<773>; using type =
05026
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<771»; }; // NOLINT
           template<> struct ConwayPolynomial<773, 6> { using ZPZ = aerobus::zpz<773>;
                                                                                                 using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<91>, ZPZV<3>, ZPZV<581>, ZPZV<2»; };
                                                                                            // NOLINT
05028
          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<771»; }; // NOLINT
05029
           template<> struct ConwayPolynomial<773, 8> { using ZPZ = aerobus::zpz<773>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<484>, ZPZV<94>, ZPZV<693>, ZPZV<2»; };
           template<> struct ConwayPolynomial<773, 9> { using ZPZ = aerobus::zpz<773>; using type =
05030
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<216>, ZPZV<574>, ZPZV<771»;
       }; // NOLINT
           template<> struct ConwayPolynomial<787, 1> { using ZPZ = aerobus::zpz<787>; using type =
05031
      POLYV<ZPZV<1>, ZPZV<785»; }; // NOLINT
           template<> struct ConwayPolynomial<787, 2> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
05033
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785»; }; // NOLINT template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type =
05034
      POLYVCZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
05035
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<785»; }; // NOLINT
05036
           template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<98>, ZPZV<512>, ZPZV<606>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<787, 7> { using ZPZ = aerobus::zpz<787>; using type =
05037
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<785»; }; // NOLINT
           template<> struct ConwayPolynomial<787, 8> { using ZPZ = aerobus::zpz<787>; using type =
05038
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<612>, ZPZV<26>, ZPZV<715>, ZPZV<2w; };
05039
          template<> struct ConwayPolynomial<787, 9> { using ZPZ = aerobus::zpz<787>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<480>, ZPZV<573>, ZPZV<785»;
       }; // NOLINT
05040
           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 =
05041
       POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
05042
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type =
05043
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<717>, ZPZV<2»; };
                                                                       // NOLINT
           template<> struct ConwayPolynomial<797, 5> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<795»; }; // NOLINT
05045
          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 =
05046
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<795»; };
            template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2»; }; //
       NOLINT
05048
           template<> struct ConwayPolynomial<797, 9> { using ZPZ = aerobus::zpz<797>; using type =
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<240>, ZPZV<599>, ZPZV<795»;
       }; // NOLINT
05049
            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 =
05050
       POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
05051
       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<1>, ZPZV<806»; }; // NOLINT template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
       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</pre>
05053
           template<> struct ConwayPolynomial<809, 6> { using ZPZ = aerobus::zpz<809>; using type =
05054
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<562>, ZPZV<75>, ZPZV<43>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<809,
                                                          7> { using ZPZ = aerobus::zpz<809>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<806»; }; // NOLINT
05056
           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<673>, ZPZV<3»; }; //
       NOLINT
05057
           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<0>, ZPZV<1>, ZPZV<341>, ZPZV<727-, ZPZV<806»;
       }; // NOLINT
05058
           template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
       POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
           template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
05059
       POLYV<ZPZV<1>, ZPZV<806>, ZPZV<3»; }; // NOLINT
05060
            template<> struct ConwayPolynomial<811, 3> { using ZPZ = aerobus::zpz<811>; using type =
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZV<608»; }; // NOLINT template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type =
05061
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<811, 5> { using ZPZ = aerobus::zpz<811>; using type =
05062
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<3>, ZPZY<808»; }; // NOLINT template<> struct ConwayPolynomial<811, 6> { using ZPZ = aerobus::zpz<811>; using type =
       POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<780>, ZPZV<755>, ZPZV<307>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<811, 7> { using ZPZ = aerobus::zpz<811>; using type =
05064
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<808»; }; // NOLINT
           template<> struct ConwayPolynomial<811, 8> { using ZPZ = aerobus::zpz<811>; using type =
05065
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<525>, ZPZV<3»; }; //
       template<> struct ConwayPolynomial<811, 9> { using ZPZ = aerobus::zpz<811>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<382>, ZPZV<200>, ZPZV<808*;</pre>
       }; // NOLINT
05067
           template<> struct ConwayPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
       POLYV<ZPZV<1>, ZPZV<819»; }; // NOLINT
           template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
05068
       POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<821, 3> { using ZPZ = aerobus::zpz<821>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<819»; }; // NOLINT template<> struct ConwayPolynomial<821, 4> { using ZPZ = aerobus::zpz<821>; using type =
05070
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<21>, ZPZV<21>, ZPZV<21>, ZPZV<22, ZPZV<22; }; // NOLINT template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
05071
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<819»; }; // NOLINT
           template<> struct ConwayPolynomial<821, 6> { using ZPZ = aerobus::zpz<821>; using type =
05072
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<160>, ZPZV<130>, ZPZV<803>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type =
05073
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<819»; }; // NOLINT template<> struct ConwayPolynomial<821, 8> { using ZPZ = aerobus::zpz<821>; using type =
05074
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<626>, ZPZV<556>, ZPZV<589>, ZPZV<2»; }; //
05075
           template<> struct ConwayPolynomial<821, 9> { using ZPZ = aerobus::zpz<821>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<650>, ZPZV<557>, ZPZV<819»;
       }; // NOLINT
           template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
05076
       POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
            template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
       POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; }; // NOLINT
05078
           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 =
05079
       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
05081
           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 =
05082
       POLYV<ZPZV<1>, 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 =
05083
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<451>, ZPZV<437>, ZPZV<31>, ZPZV<3»; };
       NOLINT
05084
           template<> struct ConwayPolynomial<823, 9> { using ZPZ = aerobus::zpz<823>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<64>, ZPZV<740>, ZPZV<609>, ZPZV<820»;
       }; // NOLINT
```

```
05085
                      template<> struct ConwayPolynomial<827, 1> { using ZPZ = aerobus::zpz<827>; using type =
             POLYV<ZPZV<1>, ZPZV<825»; }; // NOLINT
05086
                     template<> struct ConwayPolynomial<827, 2> { using ZPZ = aerobus::zpz<827>; using type =
             POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT
05087
                     template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
            POLYV<ZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<625, ZPZV<625, ZPZV<825»; }; // NOLINT template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2»; };
                                                                                                                                        // NOLINT
                     template<> struct ConwayPolynomial<827, 5> { using ZPZ = aerobus::zpz<827>; using type =
05089
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<825»; }; // NOLINT
05090
                     template<> struct ConwayPolynomial<827, 6> { using ZPZ = aerobus::zpz<827>; using type =
              \verb"POLYV<2PZV<1>, \ ZPZV<6>, \ ZPZV<9>, \ ZPZV<685>, \ ZPZV<601>, \ ZPZV<691>, \ ZPZV<2»; \ \}; \ // \ NOLINT \ A poly 
                     template<> struct ConwayPolynomial<827,
                                                                                                          7> { using ZPZ = aerobus::zpz<827>; using type
05091
             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5 , ZPZV<5
05092
                    template<> struct ConwayPolynomial<827, 8> { using ZPZ = aerobus::zpz<827>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<812>, ZPZV<79>, ZPZV<32>, ZPZV<2»; };
             NOT.TNT
             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<177>, ZPZV<372>, ZPZV<825»;
05093
05094
                     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 =
05095
             POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; }; // NOLINT
                      template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
05096
             POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<5
05097
             05098
                     template<> struct ConwayPolynomial<829, 5> { using ZPZ = aerobus::zpz<829>; using type =
             POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<7>, ZPZY<82; }; // NOLINT template<> struct ConwayPolynomial<829, 6> { using ZPZ = aerobus::zpz<829>; using type =
05099
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<476>, ZPZV<817>, ZPZV<2»; };
                                                                                                                                                                                     // NOLINT
                     template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<827»; }; // NOLINT
                     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<3>, ZPZV<621>, ZPZV<552>, ZPZV<827»;
             }; // NOLINT
05103
                     template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
             POLYV<ZPZV<1>, ZPZV<828»; }; // NOLINT
                     template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
05104
             POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; }; // NOLINT
                      template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; };
                                                                                                                  // NOLINT
05106
                    template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
05107
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<828»; }; // NOLINT
05108
                     template<> struct ConwayPolynomial<839, 6> { using ZPZ = aerobus::zpz<839>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<370>, ZPZV<537>, ZPZV<23>, ZPZV<11»; }; // NOLINI
05109
                     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<7>, ZPZV<828»; }; // NOLINT template<> struct ConwayPolynomial<839, 8> { using ZPZ = aerobus::zpz<839>; using type =
05110
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<553>, ZPZV<779>, ZPZV<329>, ZPZV<11»; }; //
05111
                    template<> struct ConwayPolynomial<839, 9> { using ZPZ = aerobus::zpz<839>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<349>, ZPZV<206>, ZPZV<828*;
             }; // NOLINT
05112
                     template<> struct ConwayPolynomial<853, 1> { using ZPZ = aerobus::zpz<853>; using type =
             POLYV<ZPZV<1>, ZPZV<851»; }; // NOLINT
05113
                      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 =
05114
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851»; }; // NOLINT
template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
05115
05116
            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 =
05117
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<276>, ZPZV<194>, ZPZV<512>, ZPZV<2»; }; // NOLINT
05118
                     template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type =
             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<45, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 8> { using ZPZ = aerobus::zpz<853>; using type =
05119
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<544>, ZPZV<846>, ZPZV<118>, ZPZV<2»; }; //
05120
                     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<821>, ZPZV<851»;
             }; // NOLINT
05121
                      template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
             POLYV<ZPZV<1>, ZPZV<854»; };
                                                                           // NOLINT
                      template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
             POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; }; // NOLINT
                    template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
05123
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; }; // NOLINT template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<528>, ZPZV<3»; };
               template<> struct ConwayPolynomial<857, 5> { using ZPZ = aerobus::zpz<857>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<854»; }; // NOLINT
               template<> struct ConwayPolynomial<857, 6> { using ZPZ = aerobus::zpz<857>; using type =
05126
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<824>, ZPZV<65>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::zpz<857>; using type
05127
        POLYV<ZPZV<1>, 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
05128
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<611>, ZPZV<552>, ZPZV<494>, ZPZV<3»; }; //
05129
              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<0>, ZPZV<0>, ZPZV<308>, ZPZV<719>, ZPZV<854»;
        }; // NOLINT
   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 =
0.5131
        POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
05132
         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
05134
               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 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
05135
               template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpz<859>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<857»; };
              template<> struct ConwayPolynomial<859, 8> { using ZPZ = aerobus::zpz<859>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<52>, ZPZV<446>, ZPZV<672>, ZPZV<2»; }; //
        NOLINT
              template<> struct ConwayPolynomial<859, 9> { using ZPZ = aerobus::zpz<859>; using type =
05138
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<648>, ZPZV<845>, ZPZV<857»;
         }; // NOLINT
              template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
05139
        POLYV<ZPZV<1>, ZPZV<858»; }; // NOLINT
               template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
05140
        POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<858»; }; // NOLINT template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
05142
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<863, 5> { using ZPZ = aerobus::zpz<863>; using type =
0.5143
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<858»; }; // NOLINT
05144
               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
05145
               template<> struct ConwayPolynomial<863, 7> { using ZPZ = aerobus::zpz<863>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<858»; }; // NOLINT template<> struct ConwayPolynomial<863, 8> { using ZPZ = aerobus::zpz<863>; using type =
05146
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<576>, ZPZV<576>, ZPZV<849>, ZPZV<58; }; //
         NOLINT
05147
               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<381>, ZPZV<4858»; };
         // NOLINT
05148
               template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
        POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
05149
         POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875»; }; // NOLINT template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type =
05151
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<877, 5> { using ZPZ = aerobus::zpz<877>; using type =
05152
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<875»; }; // NOLINT
               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<2»; }; // NOLINT
05154
              template<> struct ConwayPolynomial<877, 7> { using ZPZ = aerobus::zpz<877>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<875»; }; // NOLINT template<> struct ConwayPolynomial<877, 8> { using ZPZ = aerobus::zpz<877>; using type =
05155
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<767>, ZPZV<319>, ZPZV<347>, ZPZV
         NOLINT
               template<> struct ConwayPolynomial<877, 9> { using ZPZ = aerobus::zpz<877>; using type =
05156
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<770>, ZPZV<278>, ZPZV<875»;
         }; // NOLINT
05157
               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 =
        POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT
05159
               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 =
05160
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<47>, ZPZV<27, ZPZV<28; }; // NOLINT template<> struct ConwayPolynomial<881, 5> { using ZPZ = aerobus::zpz<881>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<878»; }; // NOLINT
05162
              template<> struct ConwayPolynomial<881, 6> { using ZPZ = aerobus::zpz<881>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<419>, ZPZV<231>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<881, 7> { using ZPZ = aerobus::zpz<881>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<878»; }; // NOLINT
0.5163
```

```
template<> struct ConwayPolynomial<881, 8> { using ZPZ = aerobus::zpz<881>; using type =
            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<635>, ZPZV<490>, ZPZV<561>, ZPZV<3»; };
            NOLINT
05165
                    template<> struct ConwayPolynomial<881, 9> { using ZPZ = aerobus::zpz<881>; using type =
            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<587>, ZPZV<510>, ZPZV<878»;
            }; // NOLINT
05166
                      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 =
05167
           POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<883, 3> { using ZPZ = aerobus::zpz<883>; using type =
05168
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<881»; }; // NOLINT template<> struct ConwayPolynomial<883, 4> { using ZPZ = aerobus::zpz<883>; using type =
05169
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2»; }; // NOLINT
05170
                    template<> struct ConwayPolynomial<883, 5> { using ZPZ = aerobus::zpz<883>; using type =
             \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<14>, \verb"ZPZV<881"; \verb"]; $ // \verb"NOLINT" | NOLINT" | NOLINT"
05171
                     template<> struct ConwayPolynomial<883, 6> { using ZPZ = aerobus::zpz<883>; using type =
            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<879>, ZPZV<875, ZPZV<871>, ZPZV<2»; ; // NOLINT template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<881»; };
                     template<> struct ConwayPolynomial<883, 8> { using ZPZ = aerobus::zpz<883>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<740>, ZPZV<762>, ZPZV<768>, ZPZV<29*; }; //
            NOLINT
            template<> struct ConwayPolynomial<883, 9> { using ZPZ = aerobus::zpz<883>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<360>, ZPZV<557>, ZPZV<881»;</pre>
0.5174
            }; // NOLINT
                    template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
05175
            POLYV<ZPZV<1>, ZPZV<882»; }; // NOLINT
05176
                     template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
            POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; }; // NOLINT
                     template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
05177
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<882»; }; // NOLINT
                     template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
05178
            05179
                     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
05180
05181
                     template<> struct ConwayPolynomial<887, 7> { using ZPZ = aerobus::zpz<887>; using type
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8 , ZPZV<8
05182
                    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, }; //
            NOLINT
05183
                    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<4>, ZPZV<727>, ZPZV<345>, ZPZV<882»;
            }; // NOLINT
05184
                    template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
            POLYV<ZPZV<1>, ZPZV<905»; }; // NOLINT
                     template<> struct ConwayPolynomial<907, 2> { using ZPZ = aerobus::zpz<907>; using type =
05185
            POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2»; }; // NOLINT
05186
                     template<> struct ConwayPolynomial<907, 3> { using ZPZ = aerobus::zpz<907>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<905»; }; // NOLINT
                     template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
05187
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
05188
            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 =
            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<626>, ZPZV<752>, ZPZV<266>, ZPZV<2»; }; // NOLINT
                     template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type =
05190
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<905»; }; // NOLIN template<> struct ConwayPolynomial<907, 8> { using ZPZ = aerobus::zpz<907>; using type =
05191
            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<584>, ZPZV<518>, ZPZV<811>, ZPZV<2»; }; //
            NOLINT
                     template<> struct ConwayPolynomial<907, 9> { using ZPZ = aerobus::zpz<907>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
            }; // NOLINT
05193
                     template<> struct ConwayPolynomial<911, 1> { using ZPZ = aerobus::zpz<911>; using type =
            POLYV<ZPZV<1>, ZPZV<894»; }; // NOLINT
                     template<> struct ConwayPolynomial<911, 2> { using ZPZ = aerobus::zpz<911>; using type =
05194
            POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; }; // NOLINT
                     template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
05195
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; }; // NOLINT
                     template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
05196
            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
05197
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<894»; }; // NOLINT
                     template<> struct ConwayPolynomial<911, 6> { using ZPZ = aerobus::zpz<911>; using type =
05198
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<172>, ZPZV<683>, ZPZV<19>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
05199
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<894»; }; // NOLINT template<> struct ConwayPolynomial<911, 8> { using ZPZ = aerobus::zpz<911>; using type =
05200
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<590>, ZPZV<168>, ZPZV<17»; }; //
05201
                    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<0>, ZPZV<0>, ZPZV<679>, ZPZV<116>, ZPZV<894»;
            }; // NOLINT
05202
                    template<> struct ConwayPolynomial<919. 1> { using ZPZ = aerobus::zpz<919>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<912»; };
           template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
05204
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912»; }; // NOLINT template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
05205
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<919, 5> { using ZPZ = aerobus::zpz<919>; using type =
05206
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<912»; }; // NOLINT
      template<> struct ConwayPolynomial<919, 6> { using ZPZ = aerobus::zpz<919>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<312>, ZPZV<817>, ZPZV<113>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<919, 7> { using ZPZ = aerobus::zpz<919>; using type =
05207
05208
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<912»; };
           template<> struct ConwayPolynomial<919, 8> { using ZPZ = aerobus::zpz<919>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<202>, ZPZV<504>, ZPZV<5»; }; //
           template<> struct ConwayPolynomial<919, 9> { using ZPZ = aerobus::zpz<919>; using type =
05210
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<41>, ZPZV<41>, ZPZV<623>, ZPZV<912»;
       }; // NOLINT
           template<> struct ConwayPolynomial<929, 1> { using ZPZ = aerobus::zpz<929>; using type =
       POLYV<ZPZV<1>, ZPZV<926»; }; // NOLINT
           template<> struct ConwayPolynomial<929, 2> { using ZPZ = aerobus::zpz<929>; using type =
05212
      POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
05213
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<926»; }; // NOLINT template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3»; }; // NOLINT
05215
           template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
      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 =
05216
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<805>, ZPZV<86>, ZPZV<86>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type
05217
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<926»; }; //
05218
          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»; }; //
       NOLINT
      template<> struct ConwayPolynomial<929, 9> { using ZPZ = aerobus::zpz<929>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<481>, ZPZV<199>, ZPZV<926»;
05219
       }; // NOLINT
           template<> struct ConwayPolynomial<937, 1> { using ZPZ = aerobus::zpz<937>; using type =
05220
      POLYV<ZPZV<1>, ZPZV<932»; }; // NOLINT
           template<> struct ConwayPolynomial<937, 2> { using ZPZ = aerobus::zpz<937>; using type =
05221
       POLYV<ZPZV<1>. ZPZV<934>. ZPZV<5»: 1: // NOLINT
           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 =
05223
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<23>, ZPZV<585>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
05224
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<932»; }; // NOLINT
           template<> struct ConwayPolynomial<937, 6> { using ZPZ = aerobus::zpz<937>; using type =
05225
      POLYV<2PZV<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 =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<932»; };
05227
           template<> struct ConwayPolynomial<937, 8> { using ZPZ = aerobus::zpz<937>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>>, ZPZV<658>, ZPZV<26>, ZPZV<53>, ZPZV<5»; };
       NOLINT
05228
           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<28>, ZPZV<533>, ZPZV<483>, ZPZV<932»;
       }; // NOLINT
05229
           template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<939»; }; // NOLINT
           template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
05230
      POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type =
05232
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<941, 5> { using ZPZ = aerobus::zpz<941>; using type =
05233
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<939»; }; // NOLINT
05234
           template<> struct ConwayPolynomial<941, 6> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<459>, ZPZV<694>, ZPZV<538>, ZPZV<2»; }; // NOLINT
05235
          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<4>, ZPZV<4>, ZPZV<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 8> { using ZPZ = aerobus::zpz<941>; using type =
05236
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<675>, ZPZV<590>, ZPZV<59), ZPZV<2»; }; //
           template<> struct ConwayPolynomial<941, 9> { using ZPZ = aerobus::zpz<941>; using type
05237
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708>, ZPZV<197>, ZPZV<939»;
       }; // NOLINT
05238
           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 =
      POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2»; }; // NOLINT
05240
          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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2»; }; // NOLINT
0.5241
```

```
05242
                  template<> struct ConwayPolynomial<947, 5> { using ZPZ = aerobus::zpz<947>; using type =
          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<22; }; // NOLINT template<> struct ConwayPolynomial<947, 7> { using ZPZ = aerobus::zpz<947>; using type
05244
          POLYV<ZPZV<1>, 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 =
           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<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26</pre>
05246
           }; // NOLINT
05247
                  template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<950»; }; // NOLINT
05248
                  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 =
05249
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950»; }; // NOLINT
                  template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<865>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<953, 5> { using ZPZ = aerobus::zpz<953>; using type =
          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 =
05252
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<507, ZPZV<507, ZPZV<730>, ZPZV<730>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<953, 7> { using ZPZ = aerobus::zpz<953>; using type
05253
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
                 template<> struct ConwayPolynomial<953, 8> { using ZPZ = aerobus::zpz<953>; using type =
05254
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<579>, ZPZV<658>, ZPZV<108>, ZPZV<3»; };
           NOLINT
05255
                 template<> struct ConwayPolynomial<953, 9> { using ZPZ = aerobus::zpz<953>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<819>, ZPZV<316>, ZPZV<950»;
           }; // NOLINT
05256
                 template<> struct ConwayPolynomial<967, 1> { using ZPZ = aerobus::zpz<967>; using type =
          POLYV<ZPZV<1>, ZPZV<962»; }; // NOLINT
05257
                  template<> struct ConwayPolynomial<967, 2> { using ZPZ = aerobus::zpz<967>; using type =
          POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5»; }; // NOLINT
          template<> struct ConwayPolynomial</pr>
Struct ConwayPolynomial
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962»; }; // NOLINT</pre>
05258
05259
                  template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<967, 5> { using ZPZ = aerobus::zpz<967>; using type =
05260
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 6> { using ZPZ = aerobus::zpz<967>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<84>, ZPZV<831>, ZPZV<5»; }; // NOLINT
05261
                  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<9>, ZPZV<9>, ZPZV<962»; };
05263
                 template<> struct ConwayPolynomial<967, 8> { using ZPZ = aerobus::zpz<967>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<840>, ZPZV<502>, ZPZV<136>, ZPZV<5»; }; //
           NOLINT
                  template<> struct ConwayPolynomial<967, 9> { using ZPZ = aerobus::zpz<967>; using type =
05264
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<512>, ZPZV<783>, ZPZV<962»;
           }; // NOLINT
05265
                  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 =
05266
          POLYV<ZPZV<1>, ZPZV<970>, ZPZV<6»; }; // NOLINT
                  template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<965»; }; // NOLINT template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
05268
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<527>, ZPZV<6»; }; // NOLINT
                 template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
05269
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<965»; }; // NOLINT
05270
                  template<> struct ConwayPolynomial<971, 6> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<970>, ZPZV<729>, ZPZV<718>, ZPZV<6»; }; // NOLINT
05271
                 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»; }; // NOLINT template<> struct ConwayPolynomial<971, 8> { using ZPZ = aerobus::zpz<971>; using type =
05272
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<281>, ZPZV<206>, ZPZV<6»; }; //
           NOLINT
                  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
05274
                  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 =
05275
          POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<977, 3> { using ZPZ = aerobus::zpz<977>; using type =
05276
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<974»; }; // NOLINT template<> struct ConwayPolynomial<977, 4> { using ZPZ = aerobus::zpz<977>; using type =
05277
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<800>, ZPZV<3w; }; // NOLINT template<> struct ConwayPolynomial<977, 5> { using ZPZ = aerobus::zpz<977>; using type =
05278
          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 =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<729>, ZPZV<830>, ZPZV<753>, ZPZV<3»; }; // NOLINT
05280
                 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<0>, ZPZV<0>, ZPZV<7, 
05281
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<855>, ZPZV<807>, ZPZV<77>, ZPZV<3»; };
05282
          template<> struct ConwayPolynomial<977, 9> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<740>, ZPZV<974»;
      }; // NOLINT
05283
           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 =
05284
      POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; }; // NOLINT
05285
          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 =
05286
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5»; };
                                                                     // NOLINT
           template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<978»; }; // NOLINT
05288
          template<> struct ConwayPolynomial<983, 6> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<849>, ZPZV<296>, ZPZV<288, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type
05289
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<978»; }; // NOLINT
05290
          template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<53»; }; //
      NOLINT
05291
          template<> struct ConwayPolynomial<983, 9> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<87>, ZPZV<87>, ZPZV<87>, ZPZV<87>, ZPZV<87>, ZPZV<87>, ZPZV<87
      }; // NOLINT
           template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<985»; }; // NOLINT
          template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
05293
      POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; }; // NOLINT
          template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
05294
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985»; }; // NOLINT template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
05295
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6»; }; // NOLINT
05296
          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 =
05297
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<637>, ZPZV<278>, ZPZV<6**; }; // NOLINT 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<0>, ZPZV<7>, ZPZV<985»; };
          template<> struct ConwayPolynomial<991, 8> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<941>, ZPZV<786>, ZPZV<234>, ZPZV<6»; }; //
      NOLINT
05300
          template<> struct ConwayPolynomial<991, 9> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9, ZPZV<466>, ZPZV<266>, ZPZV<222>, ZPZV<985»;
      }; // NOLINT
05301
          template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<990»; }; // NOLINT
          template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
05302
      POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; }; // NOLINT
          template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
05303
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990»; }; // NOLINT
           template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
      05305
          template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10, ZPZV<990»; }; // NOLINT
           template<> struct ConwayPolynomial<997, 6> { using ZPZ = aerobus::zpz<997>; using type =
05306
      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
05307
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, 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 =
05308
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<473>, ZPZV<473>, ZPZV<41>, 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<0>, ZPZV<39>, ZPZV<39>, ZPZV<732>, ZPZV<616>, ZPZV<990»;
       }; // NOLINT
05310 #endif // DO_NOT_DOCUMENT
05311 } // namespace aerobus
05312 #endif // AEROBUS_CONWAY_IMPORTS
05313
05314 #endif // __INC_AEROBUS__ // NOLINT
```

# **Chapter 10**

# **Examples**

### 10.1 QuotientRing

inject a 'constant' in quotient ring

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

**Template Parameters** 

x a 'constant' from Ring point of view

### 10.2 type\_list

A list of types <int, double, float>

A list of types <int, double, float>

**Template Parameters** 

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

## 10.3 i32::template

inject a native constant

inject a native constant

**Template Parameters** 

x inject\_constant\_2<2> -> i32::template val<2>

186 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

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

v1	a value in i32
v2	a value in i32

10.11 i32::gcd\_t 187

### 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

188 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 189

### 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>

# 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>	

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

### 10.22 i64::lt\_v

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

#### **Template Parameters**

v1	: an element of aerobus::i64::val
v2	: an element of aerobus::i64::val <i64::val <1="">, i64::val &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 191

### 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>	

### 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>>

192 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  $e \rightarrow 2.718...$ 

194 Examples

# Index

```
abs t
                                                              pq64, 30
                                                              q32, 30
     aerobus, 21
add t
                                                              q64, 30
    aerobus::i32, 53
                                                              sin, 30
    aerobus::i64, 57
                                                              sinh, 30
    aerobus::polynomial < Ring >, 63
                                                              SQRT2_fraction, 31
    aerobus::Quotient < Ring, X >, 70
                                                              SQRT3 fraction, 31
    aerobus::zpz, 93
                                                              stirling_signed_t, 31
addfractions_t
                                                              stirling_unsigned_t, 31
    aerobus, 21
                                                              tan, 31
aerobus, 17
                                                              tanh, 32
    abs t, 21
                                                              taylor, 32
    addfractions_t, 21
                                                              vadd_t, 32
                                                              vmul t, 33
    aligned_malloc, 33
    alternate t, 21
                                                         aerobus::ContinuedFraction < a0 >, 46
    alternate_v, 33
                                                              type, 47
                                                              val, 47
    asin, 22
    asinh, 22
                                                         aerobus::ContinuedFraction < a0, rest... >, 47
    atan, 22
                                                              type, 48
    atanh, 23
                                                              val, 48
                                                         aerobus::ContinuedFraction < values >, 46
    bell t, 23
    bernoulli t, 23
                                                         aerobus::ConwayPolynomial, 48
    bernoulli v, 34
                                                         aerobus::Embed< i32, i64 >, 49
    combination t, 23
                                                              type, 49
                                                         aerobus::Embed< polynomial< Small >, polynomial<
    combination_v, 34
    cos, 24
                                                                  Large >>, 49
    cosh, 24
                                                              type, 49
    E_fraction, 24
                                                         aerobus::Embed< q32, q64 >, 50
    embed_int_poly_in_fractions_t, 24
                                                              type, 50
    exp, 24
                                                         aerobus::Embed< Quotient< Ring, X >, Ring >, 50
    expm1, 25
                                                         aerobus::Embed< Ring, FractionField< Ring >>, 51
    factorial_t, 25
    factorial_v, 34
                                                              type, 51
    field, 33
                                                         aerobus::Embed< Small, Large, E >, 48
    fpq32, 25
                                                         aerobus::Embed< zpz< x>, i32>, 51
    fpq64, 25
                                                              type, 51
    FractionField, 25
                                                         aerobus::i32, 52
                                                              add t, 53
    gcd t, 26
    geometric_sum, 26
                                                              div_t, 53
    Inp1, 26
                                                              eq_t, 53
    make_frac_polynomial_t, 26
                                                              eq_v, 55
    make int polynomial t, 28
                                                              gcd t, 53
    make q32 t, 28
                                                              gt t, 53
    make_q64_t, 28
                                                              inject constant t, 53
    makefraction t, 28
                                                              inject ring t, 53
    mulfractions t, 29
                                                              inner type, 53
    pi64, 29
                                                              is_euclidean_domain, 55
    PI_fraction, 29
                                                              is_field, 55
    pow_t, 29
                                                              It_t, 53
```

mod_t, 54	chebyshev_U, 40
mul_t, 54	hermite_kind, 42
one, 54	hermite_phys, 41
pos_t, 54	hermite_prob, 41
pos_v, <u>5</u> 5	laguerre, 41
sub_t, 54	legendre, 42
zero, 54	physicist, 42
aerobus::i32::val< x >, 78	probabilist, 42
enclosing_type, 79	aerobus::polynomial < Ring >, 61
eval, 79	add_t, 63
get, 79	derive_t, 63
is_zero_t, 79	div_t, 63
to_string, 80	eq_t, 63
v, 80	gcd_t, 64
aerobus::i64, 55	gt_t, 64
add_t, 57	inject_constant_t, 64
div_t, 57	inject_ring_t, 64 is_euclidean_domain, 67
eq_t, 57	is_field, 67
eq_v, 59 gcd_t, 57	It_t, 64
gcd_t, 57 gt_t, 57	mod_t, 65
gt_v, 59	monomial_t, 65
inject_constant_t, 57	mul t, 65
inject_ring_t, 57	one, 66
inner_type, 57	pos_t, 66
is_euclidean_domain, 60	pos_v, 67
is_field, 60	simplify_t, 66
It_t, 58	sub_t, 66
lt_v, 60	X, 67
/ mod_t, 58	zero, 67
mul_t, 58	aerobus::polynomial < Ring >::val < coeffN >, 89
one, 58	aN, 90
pos_t, 58	coeff_at_t, 90
pos_v, 60	degree, 91
sub_t, 58	enclosing_type, 90
zero, 58	eval, 91
aerobus::i64::val< x >, 80	is_zero_t, 90
enclosing_type, 81	is_zero_v, 91
eval, 81	ring_type, 90
get, 82	strip, 90
inner_type, 81	to_string, 91
is_zero_t, 81	aerobus::polynomial< Ring >::val< coeffN >::coeff_at<
to_string, 82	index, E >, 45
v, 82	aerobus::polynomial < Ring >::val < coeffN >::coeff_at <
aerobus::internal, 35	index, std::enable_if_t<(index< 0     index >
index_sequence_reverse, 38	0)>>, 45
is_instantiation_of_v, 38	type, 45
make_index_sequence_reverse, 38	aerobus::polynomial< Ring >::val< coeffN >::coeff_at<
type_at_t, 38	index, std::enable_if_t<(index==0)>>, 46
aerobus::is_prime< n >, 60	type, 46
value, 61	aerobus::polynomial< Ring >::val< coeffN, coeffs >,
aerobus::IsEuclideanDomain, 43	82 aN 84
aerobus::IsField, 43	aN, 84
aerobus::IsRing, 44	coeff_at_t, 84
aerobus::known_polynomials, 39 bernoulli, 39	degree, 86 enclosing_type, 84
bernstein, 40	eval, 85
chebyshev_T, 40	is_zero_t, 84
onobysnov_1, To	10_2010_1, 07

is_zero_v, 86	pos_t, 95
ring_type, 84	pos_v, 97
strip, 84	sub_t, 96
to_string, 85	zero, 96
aerobus::Quotient $<$ Ring, $X>$ , 68	aerobus:: $zpz $ :: $val < x >$ , 87
add_t, 70	enclosing_type, 88
div_t, 70	eval, 88
eq_t, 70	get, 88
eq_v, 72	is_zero_t, 88
inject_constant_t, 70	to_string, 88
inject_ring_t, 71	v, 88
is_euclidean_domain, 72	aligned_malloc
mod_t, 71	aerobus, 33
mul_t, 71	alternate_t
one, 71	aerobus, 21
pos_t, 71	alternate_v
pos_v, 72	aerobus, 33
zero, 72	aN
aerobus::Quotient $<$ Ring, X $>$ ::val $<$ V $>$ , 86	aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN $>$ , 90
raw_t, 87	aerobus::polynomial< Ring >::val< coeffN, coeffs
type, 87	>, 84
aerobus::type_list< Ts >, 74	asin
at, 75	aerobus, 22
concat, 75	asinh
insert, 75	aerobus, 22
length, 76	at
push_back, 75	aerobus::type_list< Ts >, 75
push_front, 76	atan
remove, 76	aerobus, 22
aerobus::type_list< Ts >::pop_front, 67	atanh
tail, 68	aerobus, 23
type, 68	
type, 68 aerobus::type_list< Ts >::split< index >, 73	bell_t
type, 68 aerobus::type_list< Ts >::split< index >, 73 head, 73	bell_t aerobus, 23
type, 68 aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73	bell_t aerobus, 23 bernoulli
type, 68 aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73 aerobus::type_list<>, 76	bell_t aerobus, 23 bernoulli aerobus::known_polynomials, 39
type, 68 aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73 aerobus::type_list<>, 76 concat, 77	bell_t aerobus, 23 bernoulli aerobus::known_polynomials, 39 bernoulli_t
type, 68 aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73 aerobus::type_list<>, 76 concat, 77 insert, 77	bell_t     aerobus, 23 bernoulli     aerobus::known_polynomials, 39 bernoulli_t     aerobus, 23
type, 68 aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73 aerobus::type_list<>>, 76 concat, 77 insert, 77 length, 78	bell_t     aerobus, 23 bernoulli     aerobus::known_polynomials, 39 bernoulli_t     aerobus, 23 bernoulli_v
type, 68 aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73 aerobus::type_list<>>, 76 concat, 77 insert, 77 length, 78 push_back, 77	bell_t     aerobus, 23 bernoulli     aerobus::known_polynomials, 39 bernoulli_t     aerobus, 23 bernoulli_v     aerobus, 34
type, 68 aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73 aerobus::type_list<>>, 76 concat, 77 insert, 77 length, 78 push_back, 77 push_front, 77	bell_t aerobus, 23 bernoulli aerobus::known_polynomials, 39 bernoulli_t aerobus, 23 bernoulli_v aerobus, 34 bernstein
type, 68 aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73 aerobus::type_list<>, 76 concat, 77 insert, 77 length, 78 push_back, 77 push_front, 77 aerobus::zpz, 91	bell_t     aerobus, 23 bernoulli     aerobus::known_polynomials, 39 bernoulli_t     aerobus, 23 bernoulli_v     aerobus, 34
type, 68 aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73 aerobus::type_list<>, 76 concat, 77 insert, 77 length, 78 push_back, 77 push_front, 77 aerobus::zpz, 91 add_t, 93	bell_t     aerobus, 23 bernoulli     aerobus::known_polynomials, 39 bernoulli_t     aerobus, 23 bernoulli_v     aerobus, 34 bernstein     aerobus::known_polynomials, 40
type, 68 aerobus::type_list< Ts >::split< index >, 73     head, 73     tail, 73 aerobus::type_list<>, 76     concat, 77     insert, 77     length, 78     push_back, 77     push_front, 77 aerobus::zpz, 91     add_t, 93     div_t, 93	bell_t aerobus, 23 bernoulli aerobus::known_polynomials, 39 bernoulli_t aerobus, 23 bernoulli_v aerobus, 34 bernstein aerobus::known_polynomials, 40 chebyshev_T
type, 68  aerobus::type_list< Ts >::split< index >, 73	bell_t     aerobus, 23 bernoulli     aerobus::known_polynomials, 39 bernoulli_t     aerobus, 23 bernoulli_v     aerobus, 34 bernstein     aerobus::known_polynomials, 40  chebyshev_T     aerobus::known_polynomials, 40
type, 68  aerobus::type_list< Ts >::split< index >, 73	bell_t aerobus, 23 bernoulli aerobus::known_polynomials, 39 bernoulli_t aerobus, 23 bernoulli_v aerobus, 34 bernstein aerobus::known_polynomials, 40 chebyshev_T
type, 68  aerobus::type_list< Ts >::split< index >, 73	bell_t     aerobus, 23 bernoulli     aerobus::known_polynomials, 39 bernoulli_t     aerobus, 23 bernoulli_v     aerobus, 34 bernstein     aerobus::known_polynomials, 40 chebyshev_T     aerobus::known_polynomials, 40 chebyshev_U
type, 68  aerobus::type_list< Ts >::split< index >, 73	bell_t aerobus, 23 bernoulli aerobus::known_polynomials, 39 bernoulli_t aerobus, 23 bernoulli_v aerobus, 34 bernstein aerobus::known_polynomials, 40 chebyshev_T aerobus::known_polynomials, 40 chebyshev_U aerobus::known_polynomials, 40
type, 68  aerobus::type_list< Ts >::split< index >, 73     head, 73     tail, 73  aerobus::type_list<>>, 76     concat, 77     insert, 77     length, 78     push_back, 77     push_front, 77  aerobus::zpz, 91     add_t, 93     div_t, 93     eq_t, 93     eq_t, 94     gt_t, 94     gt_t, 94     gt_v, 96	bell_t     aerobus, 23 bernoulli     aerobus::known_polynomials, 39 bernoulli_t     aerobus, 23 bernoulli_v     aerobus, 34 bernstein     aerobus::known_polynomials, 40 chebyshev_T     aerobus::known_polynomials, 40 chebyshev_U     aerobus::known_polynomials, 40 coeff_at_t
type, 68  aerobus::type_list< Ts >::split< index >, 73	bell_t
type, 68  aerobus::type_list< Ts >::split< index >, 73	bell_t     aerobus, 23 bernoulli     aerobus::known_polynomials, 39 bernoulli_t     aerobus, 23 bernoulli_v     aerobus, 34 bernstein     aerobus::known_polynomials, 40  chebyshev_T     aerobus::known_polynomials, 40 chebyshev_U     aerobus::known_polynomials, 40 coeff_at_t     aerobus::polynomial < Ring >::val < coeffN >, 90     aerobus::polynomial < Ring >::val < coeffN, coeffs
type, 68  aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73  aerobus::type_list<>, 76 concat, 77 insert, 77 length, 78 push_back, 77 push_front, 77  aerobus::zpz, 91 add_t, 93 div_t, 93 eq_t, 93 eq_t, 93 eq_v, 96 gcd_t, 94 gt_t, 94 gt_t, 94 gt_v, 96 inject_constant_t, 94 inner_type, 94 is_euclidean_domain, 97	bell_t
type, 68  aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73  aerobus::type_list<>>, 76 concat, 77 insert, 77 length, 78 push_back, 77 push_front, 77  aerobus::zpz, 91 add_t, 93 div_t, 93 eq_t, 93 eq_t, 93 eq_v, 96 gcd_t, 94 gt_t, 94 gt_t, 94 gt_v, 96 inject_constant_t, 94 inner_type, 94 is_euclidean_domain, 97 is_field, 97	bell_t
type, 68  aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73  aerobus::type_list<>>, 76 concat, 77 insert, 77 length, 78 push_back, 77 push_front, 77  aerobus::zpz, 91 add_t, 93 div_t, 93 eq_t, 93 eq_t, 93 eq_v, 96 gcd_t, 94 gt_t, 94 gt_t, 94 gt_v, 96 inject_constant_t, 94 inner_type, 94 is_euclidean_domain, 97 is_field, 97 lt_t, 94	bell_t
type, 68  aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73  aerobus::type_list<>>, 76 concat, 77 insert, 77 length, 78 push_back, 77 push_front, 77  aerobus::zpz, 91 add_t, 93 div_t, 93 eq_t, 93 eq_t, 93 eq_v, 96 gcd_t, 94 gt_t, 94 gt_v, 96 inject_constant_t, 94 inner_type, 94 is_euclidean_domain, 97 is_field, 97 lt_t, 94 lt_v, 97	bell_t
type, 68  aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73  aerobus::type_list<>>, 76 concat, 77 insert, 77 length, 78 push_back, 77 push_front, 77  aerobus::zpz, 91 add_t, 93 div_t, 93 eq_t, 93 eq_t, 93 eq_v, 96 gcd_t, 94 gt_t, 94 gt_v, 96 inject_constant_t, 94 inner_type, 94 is_euclidean_domain, 97 is_field, 97 lt_t, 94 lt_v, 97 mod_t, 95	bell_t
type, 68  aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73  aerobus::type_list<>>, 76 concat, 77 insert, 77 length, 78 push_back, 77 push_front, 77  aerobus::zpz, 91 add_t, 93 div_t, 93 eq_t, 93 eq_t, 93 eq_v, 96 gcd_t, 94 gt_t, 94 gt_v, 96 inject_constant_t, 94 inner_type, 94 is_euclidean_domain, 97 is_field, 97 lt_t, 94 lt_v, 97 mod_t, 95 mul_t, 95	bell_t
type, 68  aerobus::type_list< Ts >::split< index >, 73 head, 73 tail, 73  aerobus::type_list<>>, 76 concat, 77 insert, 77 length, 78 push_back, 77 push_front, 77  aerobus::zpz, 91 add_t, 93 div_t, 93 eq_t, 93 eq_t, 93 eq_v, 96 gcd_t, 94 gt_t, 94 gt_v, 96 inject_constant_t, 94 inner_type, 94 is_euclidean_domain, 97 is_field, 97 lt_t, 94 lt_v, 97 mod_t, 95	bell_t

	aerobus, 24	fpq64
cosh		aerobus, 25
	aerobus, 24	FractionField
degre		aerobus, 25
-	aerobus::polynomial< Ring >::val< coeffN >, 91	gcd_t
	aerobus::polynomial < Ring >::val < coeffN, coeffs	aerobus, 26
	>, 86	aerobus::i32, 53
deriv	•	aerobus::i64, 57
	aerobus::polynomial< Ring >, 63	aerobus::polynomial< Ring >, 64
div_t		aerobus:: $zpz $ , 94
	aerobus::i32, <u>53</u>	geometric_sum
	aerobus::i64, 57	aerobus, 26
	aerobus::polynomial< Ring >, 63	get
	aerobus::Quotient< Ring, X >, 70	aerobus::i32::val< x >, 79
	aerobus:: $zpz $ , 93	aerobus::i64::val< x >, 82
	шо. оздол <u>е</u> ре х р у , ос	aerobus::zpz $<$ p $>::val<$ x $>$ , 88
E_fra	action	gt_t
	aerobus, 24	aerobus::i32, 53
embe	ed_int_poly_in_fractions_t	aerobus::i64, 57
	aerobus, 24	aerobus::polynomial< Ring >, 64
enclo	sing_type	aerobus:: $zpz $ , 94
	aerobus::i32::val< x >, 79	gt_v
	aerobus:: $i64::val < x > , 81$	aerobus::i64, 59
	aerobus::polynomial< Ring >::val< coeffN >, 90	aerobus::zpz, 96
	aerobus::polynomial< Ring >::val< coeffN, coeffs	, , , , , , , , , , , , , , , , , , ,
	>, 84	head
	aerobus:: $zpz ::val < x >$ , 88	aerobus::type_list< Ts >::split< index >, 73
eq_t		hermite_kind
	aerobus::i32, 53	aerobus::known_polynomials, 42
	aerobus::i64, 57	hermite_phys
	aerobus::polynomial< Ring >, 63	aerobus::known_polynomials, 41
	aerobus::Quotient $<$ Ring, $X>$ , 70	hermite_prob
	aerobus::zpz, 93	aerobus::known_polynomials, 41
eq_v		
	aerobus::i32, 55	index_sequence_reverse
	aerobus::i64, 59	aerobus::internal, 38
	aerobus::Quotient< Ring, X >, 72	inject_constant_t
	aerobus::zpz, 96	aerobus::i32, 53
eval		aerobus::i64, 57
	aerobus::i32::val< x >, 79	aerobus::polynomial < Ring >, 64
	aerobus::i64::val< x >, 81	aerobus::Quotient< Ring, X >, 70
	aerobus::polynomial < Ring >::val < coeffN >, 91	aerobus::zpz, 94
	aerobus::polynomial< Ring >::val< coeffN, coeffs	inject_ring_t
	>, 85	aerobus::i32, 53
	aerobus::zpz $<$ p $>$ ::val $<$ x $>$ , 88	aerobus::i64, 57
exp		aerobus::polynomial < Ring >, 64
	aerobus, 24	aerobus::Quotient< Ring, X >, 71
expm		inner_type
	aerobus, 25	aerobus::i32, 53 aerobus::i64, 57
factor	rial t	
facto		aerobus::i64::val $< x >$ , 81 aerobus::zpz, 94
facto	aerobus, 25	insert
field	aerobus, 34	aerobus::type_list< Ts >, 75 aerobus::type_list<>, 77
	aerobus, 33	Introduction, 1
fpq32		is_euclidean_domain
	aerobus, 25	aerobus::i32, 55
		, · · ·

aerobus::i64, 60	monomial_t
aerobus::polynomial < Ring >, 67	aerobus::polynomial< Ring >, 65
aerobus::Quotient < Ring, X >, 72	mul_t
aerobus::zpz, 97	aerobus::i32, 54
is field	aerobus::i64, 58
aerobus::i32, 55	aerobus::polynomial< Ring >, 65
aerobus::i64, 60	aerobus::Quotient< Ring, X >, 71
aerobus::polynomial < Ring >, 67	aerobus:: $zpz $ , 95
aerobus::zpz, 97	mulfractions_t
is_instantiation_of_v	aerobus, 29
aerobus::internal, 38	
is_zero_t	one
aerobus::i32::val< x >, 79	aerobus::i32, 54
aerobus::i64::val < $x >$ , 81	aerobus::i64, 58
aerobus::polynomial< Ring >::val< coeffN >, 90	aerobus::polynomial $<$ Ring $>$ , 66
aerobus::polynomial< Ring >::val< coeffN, coeffs	aerobus::Quotient $<$ Ring, $X>$ , 71
>, 84	aerobus::zpz, 95
aerobus::zpz $<$ p $>$ ::val $<$ x $>$ , 88	
is_zero_v	physicist
aerobus::polynomial< Ring >::val< coeffN >, 91	aerobus::known_polynomials, 42
aerobus::polynomial < Ring >::val < coeffN, coeffs	pi64
>, 86	aerobus, 29
	PI_fraction
laguerre	aerobus, 29
aerobus::known_polynomials, 41	pos_t
legendre	aerobus::i32, 54
aerobus::known_polynomials, 42	aerobus::i64, 58
length	aerobus::polynomial $<$ Ring $>$ , 66
aerobus::type_list< Ts >, 76	aerobus::Quotient $<$ Ring, $X>$ , 71
aerobus::type_list<>, 78	aerobus::zpz $<$ p $>$ , 95
Inp1	pos_v
aerobus, 26	aerobus::i32, 55
lt_t	aerobus::i64, 60
aerobus::i32, 53	aerobus::polynomial $<$ Ring $>$ , 67
aerobus::i64, 58	aerobus::Quotient< Ring, X >, 72
aerobus::polynomial $<$ Ring $>$ , 64	aerobus::zpz $<$ p $>$ , 97
aerobus::zpz, 94	pow_t
lt_v	aerobus, 29
aerobus::i64, 60	pq64
aerobus::zpz, 97	aerobus, 30
	probabilist
make_frac_polynomial_t	aerobus::known_polynomials, 42
aerobus, 26	push_back
make_index_sequence_reverse	aerobus::type_list< Ts >, 75
aerobus::internal, 38	aerobus::type_list<>, 77
make_int_polynomial_t	push_front
aerobus, 28	aerobus::type_list< Ts >, 76
make_q32_t	aerobus::type_list<>, 77
aerobus, 28	00
make_q64_t	q32
aerobus, 28	aerobus, 30
makefraction_t	q64
aerobus, 28	aerobus, 30
mod_t	row t
aerobus::i32, 54	raw_t aprobus::Ouotiont < Ring Y >::val < V > 87
aerobus::i64, 58	aerobus::Quotient< Ring, X >::val< V >, 87 README.md, 99
aerobus::polynomial < Ring >, 65	remove
aerobus::Quotient< Ring, X >, 71	
aerobus::zpz, 95	aerobus::type_list< Ts >, 76

```
ring_type
                                                             aerobus::polynomial<
                                                                                     Ring
                                                                                            >::val<
                                                                                                       coeffN
     aerobus::polynomial < Ring >::val < coeffN >, 90
                                                                  >::coeff at< index, std::enable if t<(index<
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                                 0 \mid | \text{index} > 0) > , 45
                                                             aerobus::polynomial< Ring
         >, 84
                                                                                            >::val<
                                                                                                       coeffN
                                                                 >::coeff_at< index, std::enable_if_t<(index==0)>
simplify_t
                                                                  >, 46
     aerobus::polynomial < Ring >, 66
                                                             aerobus::Quotient< Ring, X >::val< V >, 87
sin
                                                             aerobus::type list< Ts >::pop front, 68
     aerobus, 30
                                                        type at t
sinh
                                                             aerobus::internal, 38
     aerobus, 30
SQRT2 fraction
     aerobus, 31
                                                             aerobus::32::val < x >, 80
SQRT3_fraction
                                                             aerobus::i64::val < x >, 82
     aerobus, 31
                                                             aerobus::zpz ::val < x >, 88
src/aerobus.h, 99
                                                        vadd t
stirling_signed_t
                                                             aerobus, 32
     aerobus, 31
                                                        val
stirling_unsigned_t
                                                             aerobus::ContinuedFraction < a0 >, 47
     aerobus, 31
                                                             aerobus::ContinuedFraction < a0, rest... >, 48
strip
                                                        value
     aerobus::polynomial < Ring >::val < coeffN >, 90
                                                             aerobus::is_prime< n >, 61
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                        vmul t
         >, 84
                                                             aerobus, 33
sub t
                                                        Χ
     aerobus::i32, 54
                                                             aerobus::polynomial < Ring >, 67
     aerobus::i64, 58
     aerobus::polynomial < Ring >, 66
                                                        zero
     aerobus::zpz, 96
                                                             aerobus::i32, 54
                                                             aerobus::i64, 58
tail
                                                             aerobus::polynomial < Ring >, 67
     aerobus::type_list< Ts >::pop_front, 68
                                                             aerobus::Quotient < Ring, X >, 72
     aerobus::type_list< Ts >::split< index >, 73
                                                             aerobus::zpz , 96
tan
     aerobus, 31
tanh
     aerobus, 32
taylor
     aerobus, 32
to_string
     aerobus::i32::val < x >, 80
     aerobus::i64::val < x >, 82
     aerobus::polynomial< Ring >::val< coeffN >, 91
     aerobus::polynomial< Ring >::val< coeffN, coeffs
     aerobus::zpz ::val < x >, 88
type
     aerobus::ContinuedFraction< a0 >, 47
     aerobus::ContinuedFraction < a0, rest... >, 48
     aerobus::Embed< i32, i64 >, 49
     aerobus::Embed<
                         polynomial<
                                          Small
                                                   >,
         polynomial< Large >>, 49
     aerobus::Embed< q32, q64 >, 50
     aerobus::Embed< Quotient< Ring, X >, Ring >,
     aerobus::Embed< Ring, FractionField< Ring >>,
     aerobus::Embed< zpz< x>, i32>, 51
```