# 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	22
6.1.2.3 alternate_t	22
6.1.2.4 asin	22
6.1.2.5 asinh	22
6.1.2.6 atan	23
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	24
6.1.2.11 cos	24
6.1.2.12 cosh	24

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	39
6.2.4.1 is_instantiation_of_v	39
6.3 aerobus::known_polynomials Namespace Reference	39
6.3.1 Detailed Description	39
6.3.2 Typedef Documentation	40
6.3.2.1 bernoulli	40
6.3.2.2 bernstein	40
6.3.2.3 chebyshev_T	40
6.3.2.4 chebyshev_U	41
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	43
6.3.3.1 hermite_kind	43
7 Concept Documentation	45
7.1 aerobus::IsEuclideanDomain Concept Reference	
7.1.1 Concept definition	
7.1.2 Detailed Description	
7.2 aerobus::IsField Concept Reference	
7.2.1 Concept definition	
7.2.2 Detailed Description	
7.3 aerobus::IsRing Concept Reference	
7.3.1 Concept definition	
7.3.2 Detailed Description	
3 Class Documentation	47
8.1 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E > Struct Template Reference	47
8.2 aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable_if_t < (index < 0  index > 0) > > Struct Template Reference	47
8.2.1 Member Typedef Documentation	47
8.2.1.1 type	47
8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference	48
8.3.1 Member Typedef Documentation	

8.3.1.1 type	48
8.4 aerobus::ContinuedFraction< values > Struct Template Reference	48
8.4.1 Detailed Description	48
8.5 aerobus::ContinuedFraction< a0 > Struct Template Reference	49
8.5.1 Detailed Description	49
8.5.2 Member Typedef Documentation	49
8.5.2.1 type	49
8.5.3 Member Data Documentation	49
8.5.3.1 val	49
8.6 aerobus::ContinuedFraction< a0, rest > Struct Template Reference	50
8.6.1 Detailed Description	50
8.6.2 Member Typedef Documentation	50
8.6.2.1 type	50
8.6.3 Member Data Documentation	51
8.6.3.1 val	51
8.7 aerobus::ConwayPolynomial Struct Reference	51
8.8 aerobus::Embed < Small, Large, E > Struct Template Reference	51
8.8.1 Detailed Description	51
8.9 aerobus::Embed $<$ i32, i64 $>$ Struct Reference	52
8.9.1 Detailed Description	52
8.9.2 Member Typedef Documentation	52
8.9.2.1 type	52
8.10 aerobus::Embed< polynomial< Small $>$ , polynomial< Large $>$ > Struct Template Reference	52
8.10.1 Detailed Description	53
8.10.2 Member Typedef Documentation	53
8.10.2.1 type	53
8.11 aerobus::Embed < q32, q64 > Struct Reference	53
8.11.1 Detailed Description	53
8.11.2 Member Typedef Documentation	54
8.11.2.1 type	54
8.12 aerobus::Embed < Quotient < Ring, X >, Ring > Struct Template Reference	54
8.12.1 Detailed Description	54
8.12.2 Member Typedef Documentation	55
8.12.2.1 type	55
8.13 aerobus::Embed < Ring, FractionField < Ring > > Struct Template Reference	55
8.13.1 Detailed Description	55
8.13.2 Member Typedef Documentation	56
8.13.2.1 type	56
8.14 aerobus::Embed $<$ zpz $<$ x $>$ , i32 $>$ Struct Template Reference	56
8.14.1 Detailed Description	56
8.14.2 Member Typedef Documentation	57
8.14.2.1 type	57

8.15 aerobus::i32 Struct Reference	57
8.15.1 Detailed Description	58
8.15.2 Member Typedef Documentation	58
8.15.2.1 add_t	58
8.15.2.2 div_t	58
8.15.2.3 eq_t	58
8.15.2.4 gcd_t	59
8.15.2.5 gt_t	59
8.15.2.6 inject_constant_t	59
8.15.2.7 inject_ring_t	59
8.15.2.8 inner_type	59
8.15.2.9 lt_t	59
8.15.2.10 mod_t	59
8.15.2.11 mul_t	59
8.15.2.12 one	60
8.15.2.13 pos_t	60
8.15.2.14 sub_t	60
8.15.2.15 zero	60
8.15.3 Member Data Documentation	60
8.15.3.1 eq_v	60
8.15.3.2 is_euclidean_domain	60
8.15.3.3 is_field	60
8.15.3.4 pos_v	61
8.16 aerobus::i64 Struct Reference	61
8.16.1 Detailed Description	62
8.16.2 Member Typedef Documentation	62
8.16.2.1 add_t	62
8.16.2.2 div_t	62
8.16.2.3 eq_t	62
8.16.2.4 gcd_t	62
8.16.2.5 gt_t	62
8.16.2.6 inject_constant_t	63
8.16.2.7 inject_ring_t	63
8.16.2.8 inner_type	63
8.16.2.9 lt_t	63
8.16.2.10 mod_t	63
8.16.2.11 mul_t	63
8.16.2.12 one	63
8.16.2.13 pos_t	64
8.16.2.14 sub_t	64
8.16.2.15 zero	64
8.16.3 Member Data Documentation	64

8.16.3.1 eq_v	. 64
8.16.3.2 gt_v	. 64
8.16.3.3 is_euclidean_domain	. 64
8.16.3.4 is_field	. 64
8.16.3.5 lt_v	. 65
8.16.3.6 pos_v	. 65
8.17 aerobus::is_prime $<$ n $>$ Struct Template Reference	. 65
8.17.1 Detailed Description	. 65
8.17.2 Member Data Documentation	. 65
8.17.2.1 value	. 65
8.18 aerobus::polynomial < Ring > Struct Template Reference	. 66
8.18.1 Detailed Description	. 67
8.18.2 Member Typedef Documentation	. 67
8.18.2.1 add_t	. 67
8.18.2.2 derive_t	. 68
8.18.2.3 div_t	. 68
8.18.2.4 eq_t	. 68
8.18.2.5 gcd_t	. 68
8.18.2.6 gt_t	. 69
8.18.2.7 inject_constant_t	. 69
8.18.2.8 inject_ring_t	. 69
8.18.2.9 lt_t	. 69
8.18.2.10 mod_t	. 70
8.18.2.11 monomial_t	. 70
8.18.2.12 mul_t	. 70
8.18.2.13 one	. 70
8.18.2.14 pos_t	. 71
8.18.2.15 simplify_t	. 71
8.18.2.16 sub_t	. 71
8.18.2.17 X	. 71
8.18.2.18 zero	. 71
8.18.3 Member Data Documentation	. 72
8.18.3.1 is_euclidean_domain	. 72
8.18.3.2 is_field	. 72
8.18.3.3 pos_v	. 72
8.19 aerobus::type_list< Ts >::pop_front Struct Reference	. 72
8.19.1 Detailed Description	. 72
8.19.2 Member Typedef Documentation	. 73
8.19.2.1 tail	. 73
8.19.2.2 type	. 73
8.20 aerobus::Quotient $<$ Ring, X $>$ Struct Template Reference	. 73
8.20.1 Detailed Description	. 74

8.20.2 Member Typedef Documentation	74
8.20.2.1 add_t	74
8.20.2.2 div_t	75
8.20.2.3 eq_t	75
8.20.2.4 inject_constant_t	75
8.20.2.5 inject_ring_t	75
8.20.2.6 mod_t	76
8.20.2.7 mul_t	76
8.20.2.8 one	76
8.20.2.9 pos_t	76
8.20.2.10 zero	77
8.20.3 Member Data Documentation	77
8.20.3.1 eq_v	77
8.20.3.2 is_euclidean_domain	77
8.20.3.3 pos_v	77
$8.21 \; aerobus:: type\_list < Ts > ::split < index > Struct \; Template \; Reference \; . \; . \; . \; . \; . \; . \; . \; . \; . \; $	78
8.21.1 Detailed Description	78
8.21.2 Member Typedef Documentation	78
8.21.2.1 head	78
8.21.2.2 tail	78
8.22 aerobus::type_list< Ts > Struct Template Reference	78
8.22.1 Detailed Description	79
8.22.2 Member Typedef Documentation	79
8.22.2.1 at	79
8.22.2.2 concat	80
8.22.2.3 insert	80
8.22.2.4 push_back	80
8.22.2.5 push_front	80
8.22.2.6 remove	81
8.22.3 Member Data Documentation	81
8.22.3.1 length	81
8.23 aerobus::type_list<> Struct Reference	81
8.23.1 Detailed Description	82
8.23.2 Member Typedef Documentation	82
8.23.2.1 concat	82
8.23.2.2 insert	82
8.23.2.3 push_back	82
8.23.2.4 push_front	82
8.23.3 Member Data Documentation	82
8.23.3.1 length	82
8.24 aerobus::i32::val< x > Struct Template Reference	82
8 24 1 Detailed Description	83

8.24.2 Member Typedef Documentation	8	3
8.24.2.1 enclosing_type	8	13
8.24.2.2 is_zero_t	8	13
8.24.3 Member Function Documentation	8	14
8.24.3.1 eval()	8	14
8.24.3.2 get()	8	34
8.24.3.3 to_string()	8	34
8.24.4 Member Data Documentation	8	34
8.24.4.1 v	8	34
8.25 aerobus::i64::val $<$ x $>$ Struct Template Reference	8	5
8.25.1 Detailed Description	8	5
8.25.2 Member Typedef Documentation	8	16
8.25.2.1 enclosing_type	8	16
8.25.2.2 inner_type	8	16
8.25.2.3 is_zero_t	8	16
8.25.3 Member Function Documentation	8	16
8.25.3.1 eval()	8	16
8.25.3.2 get()	8	16
8.25.3.3 to_string()	8	7
8.25.4 Member Data Documentation	8	7
8.25.4.1 v	8	37
8.26 aerobus::polynomial < Ring >::val < coeffN, coeffs > Struct Template Reference	8	7
8.26 aerobus::polynomial < Ring >::val < coeffN, coeffs > Struct Template Reference		
	8	88
8.26.1 Detailed Description	8	38
8.26.1 Detailed Description	8	38 38
8.26.1 Detailed Description	8 8 8	38 38 38
8.26.1 Detailed Description	8 8 8 8	38 38 38
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN  8.26.2.2 coeff_at_t  8.26.2.3 enclosing_type	8 8 8 8	38 38 38 38
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN	8 8 8 8	38 38 38 39 39
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN  8.26.2.2 coeff_at_t  8.26.2.3 enclosing_type  8.26.2.4 is_zero_t  8.26.2.5 ring_type	8 8 8 8	38 38 38 39 39
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN  8.26.2.2 coeff_at_t  8.26.2.3 enclosing_type  8.26.2.4 is_zero_t  8.26.2.5 ring_type  8.26.2.6 strip	8 8 8 8	38 38 38 39 39 39
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN  8.26.2.2 coeff_at_t  8.26.2.3 enclosing_type  8.26.2.4 is_zero_t  8.26.2.5 ring_type  8.26.2.6 strip  8.26.3 Member Function Documentation		38 38 38 39 39 39
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN  8.26.2.2 coeff_at_t  8.26.2.3 enclosing_type  8.26.2.4 is_zero_t  8.26.2.5 ring_type  8.26.2.6 strip  8.26.3 Member Function Documentation  8.26.3.1 eval()		88888888888888888888888888888888888888
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN  8.26.2.2 coeff_at_t  8.26.2.3 enclosing_type  8.26.2.4 is_zero_t  8.26.2.5 ring_type  8.26.2.6 strip  8.26.3 Member Function Documentation  8.26.3.1 eval()  8.26.3.2 to_string()		388 388 388 388 388 388 388 388 388 388
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN  8.26.2.2 coeff_at_t  8.26.2.3 enclosing_type  8.26.2.4 is_zero_t  8.26.2.5 ring_type  8.26.2.6 strip  8.26.3 Member Function Documentation  8.26.3.1 eval()  8.26.3.2 to_string()  8.26.4 Member Data Documentation		388 388 388 388 388 388 388 388 388 388
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN  8.26.2.2 coeff_at_t  8.26.2.3 enclosing_type  8.26.2.4 is_zero_t  8.26.2.5 ring_type  8.26.2.6 strip  8.26.3 Member Function Documentation  8.26.3.1 eval()  8.26.3.2 to_string()  8.26.4 Member Data Documentation  8.26.4.1 degree		38 38 38 38 38 38 38 38 38 30 30 30 30 30 30 30 30 30 30 30 30 30
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN		388 388 389 399 399 399 399 399 399 399
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN  8.26.2.2 coeff_at_t  8.26.2.3 enclosing_type  8.26.2.4 is_zero_t  8.26.2.5 ring_type  8.26.2.6 strip  8.26.3 Member Function Documentation  8.26.3.1 eval()  8.26.3.2 to_string()  8.26.4 Member Data Documentation  8.26.4.1 degree  8.26.4.2 is_zero_v  8.27 aerobus::Quotient < Ring, X >::val < V > Struct Template Reference		388 388 389 389 389 389 389 389 389 389
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN  8.26.2.2 coeff_at_t  8.26.2.3 enclosing_type  8.26.2.4 is_zero_t  8.26.2.5 ring_type  8.26.2.6 strip  8.26.3 Member Function Documentation  8.26.3.1 eval()  8.26.3.2 to_string()  8.26.4 Member Data Documentation  8.26.4.1 degree  8.26.4.2 is_zero_v  8.27 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference  8.27.1 Detailed Description		388 388 389 399 399 399 399 399 399 399
8.26.1 Detailed Description  8.26.2 Member Typedef Documentation  8.26.2.1 aN  8.26.2.2 coeff_at_t  8.26.2.3 enclosing_type  8.26.2.4 is_zero_t  8.26.2.5 ring_type  8.26.2.6 strip  8.26.3 Member Function Documentation  8.26.3.1 eval()  8.26.3.2 to_string()  8.26.4 Member Data Documentation  8.26.4.1 degree  8.26.4.2 is_zero_v  8.27 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference  8.27.1 Detailed Description  8.27.2 Member Typedef Documentation		388 388 389 399 399 399 399 399 399 399

8.28.1 Detailed Description	92
8.28.2 Member Typedef Documentation	92
8.28.2.1 enclosing_type	92
8.28.2.2 is_zero_t	93
8.28.3 Member Function Documentation	93
8.28.3.1 eval()	93
8.28.3.2 get()	93
8.28.3.3 to_string()	93
8.28.4 Member Data Documentation	93
8.28.4.1 is_zero_v	93
8.28.4.2 v	94
8.29 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference	94
8.29.1 Detailed Description	94
8.29.2 Member Typedef Documentation	95
8.29.2.1 aN	95
8.29.2.2 coeff_at_t	95
8.29.2.3 enclosing_type	95
8.29.2.4 is_zero_t	95
8.29.2.5 ring_type	95
8.29.2.6 strip	95
8.29.3 Member Function Documentation	96
8.29.3.1 eval()	96
8.29.3.2 to_string()	96
8.29.4 Member Data Documentation	96
8.29.4.1 degree	96
8.29.4.2 is_zero_v	96
8.30 aerobus::zpz Struct Template Reference	96
8.30.1 Detailed Description	98
8.30.2 Member Typedef Documentation	98
8.30.2.1 add_t	98
8.30.2.2 div_t	98
8.30.2.3 eq_t	99
8.30.2.4 gcd_t	99
8.30.2.5 gt_t	99
8.30.2.6 inject_constant_t	100
8.30.2.7 inner_type	100
8.30.2.8 lt_t	100
8.30.2.9 mod_t	100
8.30.2.10 mul_t	101
8.30.2.11 one	101
8.30.2.12 pos_t	101
8.30.2.13 sub_t	101

	8.30.2.14 zero	102
	8.30.3 Member Data Documentation	102
	8.30.3.1 eq_v	102
	8.30.3.2 gt_v	102
	8.30.3.3 is_euclidean_domain	102
	8.30.3.4 is_field	102
	8.30.3.5 lt_v	103
	8.30.3.6 pos_v	103
<u>α</u> ι	File Documentation	105
9 1	9.1 README.md File Reference	
	9.2 src/aerobus.h File Reference	
	9.3 aerobus.h	
	3.5 delobus.ii	103
10	Examples	191
	10.1 QuotientRing	191
	10.2 type_list	191
	10.3 i32::template	191
	10.4 i32::add_t	192
	10.5 i32::sub_t	192
	10.6 i32::mul_t	192
	10.7 i32::div_t	192
	10.8 i32::gt_t	193
	10.9 i32::eq_t	193
	10.10 i32::eq_v	193
	10.11 i32::gcd_t	193
	10.12 i32::pos_t	194
	10.13 i32::pos_v	194
	10.14 i64::template	194
	10.15 i64::add_t	194
	10.16 i64::sub_t	195
	10.17 i64::mul_t	195
	10.18 i64::div_t	195
	10.19 i64::mod_t	195
	10.20 i64::gt_t	196
	10.21 i64::lt_t	196
	10.22 i64::lt_v	196
	10.23 i64::eq_t	196
	10.24 i64::eq_v	197
	10.25 i64::gcd_t	197
	10.26 i64::pos_t	197
	10.27 i64::pos_v	197
	10.28 polynomial	198

Index										199
10.32 E_fraction::va	վ	 	198							
10.31 PI_fraction::va	al	 	198							
10.30 FractionField		 	198							
10.29 q32::add_t		 	198							

# 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	45
aerobus::IsField	
Concept to express R is a field	45
aerobus::IsRing	
Concept to express B is a Bing	46

12 Concept Index

# **Class Index**

## 4.1 Class List

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

aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E >	47
aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable_if_t < (index < 0  index > 0)> > 47	
aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)>>	48
aerobus::ContinuedFraction < values >	
Continued fraction a0 + $\frac{1}{a_1 + \frac{1}{a_2 + \dots}}$	48
$a_1 + \frac{a_1 + a_2 + \dots}{a_2 + \dots}$ aerobus::ContinuedFraction< a0 >	
Specialization for only one coefficient, technically just 'a0'	49
aerobus::ContinuedFraction < a0, rest >	70
Specialization for multiple coefficients (strictly more than one)	50
aerobus::ConwayPolynomial	51
aerobus::Embed< Small, Large, E >	51
Embedding - struct forward declaration	51
aerobus::Embed< i32, i64 >	01
Embeds i32 into i64	52
aerobus::Embed< polynomial< Small >, polynomial< Large >>	J_
Embeds polynomial < Small > into polynomial < Large >	52
aerobus::Embed< q32, q64 >	J_
Embeds q32 into q64	53
aerobus::Embed< Quotient< Ring, X >, Ring >	00
Embeds Quotient < Ring, X >, ring	54
aerobus::Embed< Ring, FractionField< Ring > >	0.
Embeds values from Ring to its field of fractions	55
aerobus::Embed $<$ zpz $<$ x $>$ , i32 $>$	•
Embeds zpz values into i32	56
aerobus::i32	
32 bits signed integers, seen as a algebraic ring with related operations	57
aerobus::i64	•
64 bits signed integers, seen as a algebraic ring with related operations	61
aerobus::is_prime< n >	٠.
Checks if n is prime	65
aerobus::polynomial < Ring >	66
aerobus::type list< Ts >::pop front	
— · · · · —	72

14 Class Index

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	73
aerobus::type_list< Ts >::split< index >	
Splits list at index	78
aerobus::type_list< Ts >	
Empty pure template struct to handle type list	78
aerobus::type_list<>	
Specialization for empty type list	81
aerobus::i32::val< x >	
Values in i32, again represented as types	82
aerobus::i64::val< x >	
Values in i64	85
aerobus::polynomial< Ring >::val< coeffN, coeffs >	
Values (seen as types) in polynomial ring	87
aerobus::Quotient < Ring, X >::val < V >	
Projection values in the quotient ring	91
aerobus::zpz::val< x >	
Values in zpz	91
aerobus::polynomial < Ring >::val < coeffN >	
Specialization for constants	94
aerobus::zpz	
Congruence classes of integers modulo p (32 bits)	96

# File Index

- 4		 
<b>5</b> 7	FIIE	 st
J. I		 31

Here is a list of all files with brief descriptions:	
src/aerobus.h	105

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
```

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

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

```
embedding - struct forward declaration
```

struct Embed< i32, i64 >

embeds i32 into i64

struct Embed< polynomial< Small >, polynomial< Large > >

embeds polynomial<Small> into polynomial<Large>

struct Embed< q32, q64 >

embeds q32 into q64

struct Embed< Quotient< Ring, X >, Ring >

embeds Quotient<Ring, X> into Ring

struct Embed< Ring, FractionField< Ring > >

embeds values from Ring to its field of fractions

struct Embed< zpz< x >, i32 >

embeds zpz values into 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

congruence classes of integers modulo p (32 bits)

### Concepts

· concept IsRing

Concept to express R is a Ring.

• concept IsEuclideanDomain

Concept to express R is an euclidean domain.

concept IsField

Concept to express R is a field.

### **Typedefs**

```
    template < typename T, typename A, typename B >
        using gcd_t = typename internal::gcd < T >::template type < A, B >
        computes the greatest common divisor or A and B
```

• template<typename... vals>

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

adds multiple values (v1 + v2 + ... + vn) vals must have same "enclosing\_type" and "enclosing\_type" must have an add\_t binary operator

• template<typename... vals>

```
using vmul t = typename internal::vmul < vals... >::type
```

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

• template<typename val >

using abs\_t = std::conditional\_t< val::enclosing\_type::template pos\_v< val >, val, typename val::enclosing $\leftarrow$ \_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

 $\bullet \ \ \text{template}{<} \text{typename Ring} >$ 

```
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>,
  i64::inject_constant_t< q >>>
     helper type: make a fraction from numerator and denominator
• template<int32 t p, int32 t q>
  using make_q32_t = typename q32::template simplify_t< typename q32::val< i32::inject_constant_t< p >,
  i32::inject_constant_t< q >>>
     helper type: make a fraction from numerator and denominator

    template<typename Ring , typename v1 , typename v2 >

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

    template<typename Ring , typename v1 , typename v2 >

  using mulfractions t = typename FractionField < Ring >::template mul t < v1, v2 >
     helper type: multiplies two fractions
• template<typename 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 >
      e^x
• template<typename Integers , size_t deg>
  using expm1 = typename polynomial < FractionField < Integers > >::template sub t < exp < Integers, deg
  >, typename polynomial< FractionField< Integers > >::one >
• template<typename Integers , size_t deg>
  using lnp1 = taylor< Integers, internal::lnp1_coeff, deg >
      ln(1+x)
• template<typename Integers , size_t deg>
  using atan = taylor < Integers, internal::atan coeff, deg >
      \arctan(x)
• template<typename Integers , size t deg>
  using sin = taylor < Integers, internal::sin coeff, deg >
      \sin(x)
• template<typename Integers , size_t deg>
  using sinh = taylor< Integers, internal::sh_coeff, deg >
      sinh(x)
• template<typename Integers , size_t deg>
  using cosh = taylor < Integers, internal::cosh coeff, deg >
      \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
```

#### **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

### **Template Parameters**

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

### 6.1.2.3 alternate t

```
\label{template} $$ template < typename T , int k > $$ using $$ aerobus::alternate_t = typedef typename internal::alternate < T, k > ::type $$ typename T , typ
```

(-1)<sup>^</sup>k as type in T

### **Template Parameters**

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

### 6.1.2.4 asin

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

 $\arcsin(x)$  arc sinus

### **Template Parameters**

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

### 6.1.2.5 asinh

```
template<typename Integers , size_t deg> using aerobus::asinh = typedef taylor<Integers, internal::asinh_coeff, deg> \operatorname{arcsinh}(x) arc hyperbolic sinus
```

### **Template Parameters**

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

#### 6.1.2.6 atan

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

#### **Template Parameters**

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

#### 6.1.2.7 atanh

```
template<typename Integers , size_t deg> using aerobus::atanh = typedef taylor<Integers, internal::atanh_coeff, deg> \operatorname{arctanh}(x) arc hyperbolic tangent
```

#### **Template Parameters**

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

### 6.1.2.8 bell\_t

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

#### Bell numbers.

#### **Template Parameters**

ſ	Т	ring type, such as aerobus::i64
ſ	n	index

#### 6.1.2.9 bernoulli\_t

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

nth bernoulli number as type in T

#### **Template Parameters**

T	Ring type (i64)
n	

#### 6.1.2.10 combination\_t

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

computes binomial coefficient (k among n) as type

#### **Template Parameters**

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

#### 6.1.2.11 cos

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

#### **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) 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

```
template<typename v > using aerobus::embed_int_poly_in_fractions_t = typedef typename Embed< polynomial<typename v \leftrightarrow ::ring_type>, polynomial<FractionField<typename v::ring_type> >>::template type<v>
```

embed a polynomial with integers coefficients into rational coefficients polynomials

Lives in polynomial < Fraction Field < Ring >>

#### **Template Parameters**

Ring	Integers
а	value 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_ \leftrightarrow 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**

Т	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> > \cdot ::template val< typename FractionField<Ring>::template inject_constant_t<xs>...>
```

make a polynomial with coefficients in FractionField<Ring>

#### **Template Parameters**

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^n (as 'val' type in T)

#### **Template Parameters**

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

#### 6.1.2.33 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)
```

#### **Template Parameters**

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, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2
```

approximation of

#### 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**

Т	(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::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)
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

#### **Template Parameters**

```
...vals
```

#### 6.1.2.46 vmul\_t

```
template<typename... vals>
using aerobus::vmul_t = typedef 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 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 (

prime number )

#### 6.1.4 Variable Documentation

#### 6.1.4.1 alternate v

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

#### (-1)<sup>∧</sup>k as value from T

#### **Template Parameters**

```
T Ring type, aerobus::i64 for example, then result will be an int64_t
```

#### 6.1.4.2 bernoulli\_v

```
template<typename FloatType , typename T , size_t n>
constexpr FloatType aerobus::bernoulli_v = internal::bernoulli<T, n>::template value<Float←
Type> [inline], [constexpr]
```

nth bernoulli number as value in FloatType

#### **Template Parameters**

FloatType	(double or float for example)
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

#### **Template Parameters**

T	(aerobus::i64 for example)
i	

## 6.2 aerobus::internal Namespace Reference

internal implementations, subject to breaking changes without notice

struct atan\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==0 >>

#### **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 _{\bf is\_prime}< 3, i >

    struct _is_prime< 5, i >

• struct _{\bf 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)==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, I, 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, I, std::enable_if_t<(m > 0) &&(i==m)> >

· struct chebyshev_helper

    struct chebyshev helper< 1, 0, I >

    struct chebyshev_helper< 1, 1, I >

    struct chebyshev_helper< 2, 0, I >

    struct chebyshev_helper< 2, 1, I >

· struct combination

    struct combination helper

    struct combination helper
    T, 0, n >

- struct combination_helper< T, k, n, std::enable_if_t<(n >=0 &&k >(n/2) &&k > 0)> >
• struct combination helper < T, k, n, std::enable if t < (n >= 0 \&k <= (n/2) \&k > 0) > >
· struct cos_coeff
· struct cos coeff helper

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

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

· struct cosh coeff
· struct cosh_coeff_helper

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

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

    struct exp_coeff

    struct factorial

    struct factorial < T, 0 >

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

    struct FractionFieldImpl

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

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

    struct gcd

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

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

    struct geom_coeff

    struct hermite helper

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

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

- struct hermite_helper < 1, known_polynomials::hermite_kind::physicist, l >
```

struct hermite\_helper< 1, known\_polynomials::hermite\_kind::probabilist, I >

```
    struct hermite_helper< deg, known_polynomials::hermite_kind::physicist, I >

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

· struct insert h
· struct is instantiation of
• struct is instantiation of < TT, TT < Ts... >>

    struct laguerre_helper

    struct laguerre_helper< 0, I >

    struct laguerre_helper< 1, I >

• struct legendre helper

    struct legendre_helper< 0, I >

    struct legendre_helper< 1, I >

    struct Inp1_coeff

struct Inp1_coeff< T, 0 >

    struct make_taylor_impl

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

    struct 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, I >::type
     Laguerre polynomials.
• template<size_t deg, typename I = aerobus::i64>
  using hermite_prob = typename internal::hermite_helper< deg, hermite_kind::probabilist, I >::type
      Hermite polynomials - probabilist form.
• template < size_t deg, typename I = aerobus::i64>
  using hermite_phys = typename internal::hermite_helper< deg, hermite_kind::physicist, I >::type
     Hermite polynomials - physicist form.
• template < size_t i, size_t m, typename I = aerobus::i64>
  using bernstein = typename internal::bernstein helper< i, m, I >::type
      Bernstein polynomials.
• template<size_t deg, typename I = aerobus::i64>
  using legendre = typename internal::legendre helper< deg, I >::type
     Legendre polynomials.
• template < size_t deg, typename I = aerobus::i64>
```

using bernoulli = taylor< I, internal::bernoulli\_coeff< deg >::template inner, deg >

#### **Enumerations**

enum hermite\_kind { probabilist , physicist }

#### 6.3.1 Detailed Description

Bernoulli polynomials.

families of well known polynomials such as Hermite or Bernstein

#### 6.3.2 Typedef Documentation

#### 6.3.2.1 bernoulli

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

Bernoulli polynomials.

Lives in polynomial<FractionField<I>>

See also

```
See in Wikipedia
```

#### **Template Parameters**

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

#### 6.3.2.2 bernstein

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

Bernstein polynomials.

Lives in polynomial

See also

```
See in Wikipedia
```

#### Template Parameters

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

#### 6.3.2.3 chebyshev\_T

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

Chebyshev polynomials of first kind.

See also

```
See in Wikipedia
```

#### **Template Parameters**

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

#### 6.3.2.4 chebyshev\_U

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::chebyshev_U = typedef typename internal::chebyshev_helper<2,
deg, I>::type
```

Chebyshev polynomials of second kind.

Lives in polynomial

See also

See in Wikipedia

#### **Template Parameters**

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

#### 6.3.2.5 hermite\_phys

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

Hermite polynomials - physicist form.

See also

See in Wikipedia

#### **Template Parameters**

deg	degree of polynomial
-----	----------------------

#### 6.3.2.6 hermite\_prob

template<size\_t deg, typename I = aerobus::i64>

using aerobus::known\_polynomials::hermite\_prob = typedef typename internal::hermite\_helper<deg,
hermite\_kind::probabilist, I>::type

Hermite polynomials - probabilist form.

See also

```
See in Wikipedia
```

#### **Template Parameters**

```
deg degree of polynomial
```

#### 6.3.2.7 laguerre

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

Laguerre polynomials.

Lives in polynomial<FractionField<I>>

See also

```
See in Wikipedia
```

#### **Template Parameters**

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

#### 6.3.2.8 legendre

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

Legendre polynomials.

Lives in polynomial<FractionField<I>>

See also

```
See in Wikipedia
```

#### **Template Parameters**

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

## 6.3.3 Enumeration Type Documentation

#### 6.3.3.1 hermite\_kind

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

#### Enumerator

probabilist	
physicist	

## **Chapter 7**

## **Concept Documentation**

## 7.1 aerobus::IsEuclideanDomain Concept Reference

Concept to express R is an euclidean domain.

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

#### 7.1.1 Concept definition

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

#### 7.1.2 Detailed Description

Concept to express R is an euclidean domain.

## 7.2 aerobus::IsField Concept Reference

Concept to express R is a field.

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

#### 7.2.1 Concept definition

## 7.2.2 Detailed Description

Concept to express R is a field.

## 7.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring.

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

#### 7.3.1 Concept definition

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

## 7.3.2 Detailed Description

Concept to express R is a Ring.

## **Chapter 8**

## **Class Documentation**

8.1 aerobus::polynomial< Ring >::val< coeffN >::coeff\_at< index, E > Struct Template Reference

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

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

- src/aerobus.h
- 8.2 aerobus::polynomial < Ring >::val < coeffN >::coeff\_at < index, std::enable\_if\_t < (index < 0||index > 0) > > Struct Template Reference

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

#### **Public Types**

• using type = typename Ring::zero

#### 8.2.1 Member Typedef Documentation

#### 8.2.1.1 type

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

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

• src/aerobus.h

48 Class Documentation

# 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

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

#include <aerobus.h>

#### 8.4.1 Detailed Description

template<int64\_t... values> struct aerobus::ContinuedFraction< values >

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

#### **Template Parameters**

values	are
	int64_t

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

• src/aerobus.h

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

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

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

#### **Public Types**

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

#### **Static Public Attributes**

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

#### 8.5.1 Detailed Description

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

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

#### **Template Parameters**

```
a0 an integer int64_t
```

#### 8.5.2 Member Typedef Documentation

### 8.5.2.1 type

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

#### 8.5.3 Member Data Documentation

#### 8.5.3.1 val

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

#### represented value as double

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

• src/aerobus.h

50 Class Documentation

# 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>()
 reprensented 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]
```

reprensented 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

embedding - struct forward declaration

#### 8.8.1 Detailed Description

```
template<typename Small, typename Large, typename E = void> struct aerobus::Embed< Small, Large, E >
```

embedding - struct forward declaration

#### **Template Parameters**

Small	a ring which can be embedded in Large
Large	a ring in which Small can be embedded
Ε	some default type (unused – implementation related)

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

· src/aerobus.h

52 Class Documentation

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

```
embeds i32 into i64
#include <aerobus.h>
```

#### **Public Types**

```
    template < typename val >
        using type = i64::val < static_cast < int64_t > (val::v) >
        the i64 representation of val
```

#### 8.9.1 Detailed Description

embeds i32 into i64

#### 8.9.2 Member Typedef Documentation

#### 8.9.2.1 type

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

the i64 representation of val

**Template Parameters** 

```
val a value in i32
```

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

```
embeds polynomial<Small> into polynomial<Large>
#include <aerobus.h>
```

#### **Public Types**

```
• template<typename v > using type = typename at_low< v, typename internal::make_index_sequence_reverse< v::degree+1 > > ::type
```

the polynomial<Large> reprensentation of v

#### 8.10.1 Detailed Description

```
template<typename Small, typename Large> struct aerobus::Embed< polynomial< Small >, polynomial< Large > >
```

embeds polynomial<Small> into polynomial<Large>

#### **Template Parameters**

Small	a rings which can be embedded in Large
Large	a ring in which Small can be embedded

#### 8.10.2 Member Typedef Documentation

#### 8.10.2.1 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 polynomial<Large> reprensentation of v

#### **Template Parameters**

```
v a value in polynomial<Small>
```

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

• src/aerobus.h

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

embeds q32 into q64

```
#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) >
        q64 representation of v
```

#### 8.11.1 Detailed Description

embeds q32 into q64

54 Class Documentation

#### 8.11.2 Member Typedef Documentation

#### 8.11.2.1 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)>
```

q64 representation of v

**Template Parameters** 

```
v a value in q32
```

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

```
embeds Quotient<Ring, X> into Ring
```

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

### **Public Types**

```
    template < typename val >
        using type = typename val::raw_t
        Ring reprensentation of val.
```

#### 8.12.1 Detailed Description

```
template<typename Ring, typename X> struct aerobus::Embed< Quotient< Ring, X >, Ring >
```

embeds Quotient<Ring, X> into Ring

#### **Template Parameters**

Ring	a Euclidean ring
X	a value in Ring

#### 8.12.2 Member Typedef Documentation

#### 8.12.2.1 type

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

Ring reprensentation of val.

#### **Template Parameters**

```
val a value in Quotient<Ring, X>
```

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

· src/aerobus.h

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

embeds values from Ring to its field of fractions

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

#### **Public Types**

```
    template < typename v >
        using type = typename FractionField < Ring >::template val < v, typename Ring::one >
        FractionField < Ring > reprensentation of v.
```

#### 8.13.1 Detailed Description

```
\label{lem:lembd} \begin{tabular}{ll} template < typename Ring > \\ struct aerobus:: Embed < Ring, Fraction Field < Ring > > \\ \end{tabular}
```

embeds values from Ring to its field of fractions

**Template Parameters** 

Ring an integers ring, such as i32

56 Class Documentation

#### 8.13.2 Member Typedef Documentation

#### 8.13.2.1 type

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

FractionField<Ring> reprensentation of v.

#### **Template Parameters**

```
v a Ring value
```

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

• src/aerobus.h

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

embeds zpz values into i32

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

#### **Public Types**

```
    template<typename val >
        using type = i32::val< val::v >
        the i32 reprensentation of val
```

#### 8.14.1 Detailed Description

```
template<int32_t x> struct aerobus::Embed< zpz< x >, i32 >
```

embeds zpz values into i32

#### **Template Parameters**

x an integer

#### 8.14.2 Member Typedef Documentation

#### 8.14.2.1 type

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

the i32 reprensentation of val

#### **Template Parameters**

```
val a value in zpz<x>
```

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

• src/aerobus.h

#### 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
```

58 Class Documentation

#### **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

template<typename v1 , typename v2 >

#### 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
```

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

```
template<typename 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::eq_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>>  $\cdot$  i64::val<1>

**Template Parameters** 

```
v 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

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

integers are not a field

#### 8.16.3.5 lt\_v

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

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
```

# 8.17.2 Member Data Documentation

#### 8.17.2.1 value

n

```
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**

```
V
```

#### 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^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 >
        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 > >
        using div_t = val< typename Ring::template div_t< typename v1::type, typename v2::type > >
```

using mod\_t = val< typename Ring::template mod\_t< typename v1::type, typename v2::type >>

division operator

• template<typename v1, typename v2 >

```
    modulus operator
    template < typename v1 , typename v2 >
        using eq_t = typename Ring::template eq_t < typename v1::type, typename v2::type >
        equality operator (as type)
    template < typename v1 >
        using pos_t = std::true_type
        positivity operator always true
    template < auto x >
        using inject_constant_t = val < typename Ring::template inject_constant_t < x > >
        template < typename v >
        using inject_ring_t = val < v >
```

#### **Static Public Attributes**

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

## 8.20.1 Detailed Description

```
template<typename Ring, typename X> requires IsRing<Ring> struct aerobus::Quotient< Ring, X >
```

quotien rings are euclidean domain

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**

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

## 8.20.2.8 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**

#### 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 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

```
\label{template} $$ \end{template} $$ $$ \end{template} $$$ \end{template} $$ \end{template} $$ \end{template} $$$ \en
```

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

index
```

## 8.21.2 Member Typedef Documentation

## 8.21.2.1 head

```
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

Tem	plate	<b>Paran</b>	neters

#### 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** 



## 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** 

```
T
```

#### 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 >
```

#### **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<iint32_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()

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

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

# using is\_zero\_t = std::bool\_constant< x==0 > is value zero

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

#include <aerobus.h>

values in i64

```
    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()

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

#### 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
        type of coefficient at index
```

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

```
    static std::string to_string ()
    get a string representation of polynomial
```

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

evaluates polynomial seen as a function operating on ValueRing

#### **Static Public Attributes**

```
    static constexpr size_t degree = sizeof...(coeffs)
    degree of the polynomial
```

• static constexpr bool is\_zero\_v = is\_zero\_t::value

true if polynomial is constant zero

## 8.26.1 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_\leftarrow
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 \leftarrow
::value [static], [constexpr]
```

true if polynomial is constant zero

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

• src/aerobus.h

# 8.27 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

values in zpz

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

#### **Public Types**

```
    using enclosing_type = zpz
        enclosing ring type
    using is_zero_t = std::bool_constant< v==0 >
        true_type if zero
```

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

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

#### **Static Public Attributes**

```
    static constexpr int32_t v = x % p
        actual value
    static constexpr bool is_zero_v = v == 0
        true if zero
```

## 8.28.1 Detailed Description

```
template < int32_t p > template < int32_t x > struct aerobus::zpz  ::val < x > values in zpz

Template Parameters

x an integer
```

## 8.28.2 Member Typedef Documentation

## 8.28.2.1 enclosing\_type

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

#### 8.28.2.2 is\_zero\_t

```
template<int32_t p>
template<int32_t x>
using aerobus::zpz::val< x >::is_zero_t = std::bool_constant<v == 0>
true_type if zero
```

#### 8.28.3 Member Function Documentation

#### 8.28.3.1 eval()

#### 8.28.3.2 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]
```

get value as valueType

**Template Parameters** 

```
valueType an arithmetic type, such as float
```

## 8.28.3.3 to\_string()

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

a string representation

## 8.28.4 Member Data Documentation

## 8.28.4.1 is\_zero\_v

```
template<int32_t p>
template<int32_t x>
constexpr bool aerobus::zpz::val< x >::is_zero_v = v == 0 [static], [constexpr]
true if zero
```

#### 8.28.4.2 v

actual value

```
template<int32_t p>
template<iint32_t x>
constexpr int32_t aerobus::zpz::val< x >::v = x % p [static], [constexpr]
```

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>
```

# enclosing ring type

### 8.29.2.4 is\_zero\_t

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

### 8.29.2.5 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>
```

96 Class Documentation

#### 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]
```

degree

#### 8.29.4.2 is\_zero\_v

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

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

• src/aerobus.h

# 8.30 aerobus::zpz Struct Template Reference

congruence classes of integers modulo p (32 bits)

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

## Classes

struct val

values in zpz

#### **Public Types**

```
• using inner_type = int32_t
     underlying type for values
template<auto x>
  using inject_constant_t = val< static_cast< int32_t >(x)>
     injects a constant integer into zpz
using zero = val< 0 >
     zero value

    using one = val< 1 >

     one value
• 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 true iff p is prime
    static constexpr bool is_euclidean_domain = true always true
    template<typename v1, typename v2 > static constexpr bool gt_v = gt_t<v1, v2>::value strictly greater operator (booleanvalue)
```

98 Class Documentation

# 8.30.1 Detailed Description

```
template < int32_t p>
struct aerobus::zpz 
congruence classes of integers modulo p (32 bits)

if p is prime, zpz

is a field

Template Parameters

p | a integer
```

## 8.30.2 Member Typedef Documentation

## 8.30.2.1 add\_t

```
template<int32_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

100 Class Documentation

#### 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)>
```

injects a constant integer into zpz

## **Template Parameters**

```
x an integer
```

#### 8.30.2.7 inner\_type

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

underlying type for values

#### 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>
```

one value

#### 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<iint32_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
<i>v</i> 2	a value in zpz::val

102 Class Documentation

#### 8.30.2.14 zero

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

zero value

#### 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]
```

# always true

# 8.30.3.4 is\_field

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

## true iff p is prime

#### 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<iint32_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

104 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 % 3 != 0) > \{(n := 2 \&\& n := 3 \&\& 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 {
00332
         template<typename Small, typename Large, typename E = void>
00333
          struct Embed;
00334 }
        // namespace aerobus
00335
00336 namespace aerobus {
00341
         template<typename Ring, typename X>
00342
          requires IsRing<Ring>
00343
          struct Quotient {
00346
              template <typename V>
00347
              struct val {
00348
              public:
00349
                 using raw_t = V;
00350
                  using type = abs_t<typename Ring::template mod_t<V, X>>;
00351
00352
00354
              using zero = val<typename Ring::zero>;
00355
00357
              using one = val<typename Ring::one>;
00358
00362
              template<typename v1, typename v2>
00363
              using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00364
00368
              template<typename v1, typename v2>
00369
              using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00370
00374
              template<typename v1, typename v2>
00375
              using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00376
00380
              template<typename v1, typename v2>
00381
              using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00382
00386
              template<typename v1, typename v2>
00387
              using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00388
00392
              template<typename v1, typename v2> \,
00393
              static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00394
00398
              template<typename v1>
00399
              using pos_t = std::true_type;
00400
00404
              template<typename v>
              static constexpr bool pos_v = pos_t<v>::value;
00405
00406
00408
              static constexpr bool is_euclidean_domain = true;
00409
00415
              template<auto x>
00416
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00417
00423
              template<typename v>
00424
              using inject_ring_t = val<v>;
00425
          };
00426
00430
          template<typename Ring, typename X>
          struct Embed<Quotient<Ring, X>, Ring> {
00431
              template<typename val>
00434
00435
              using type = typename val::raw_t;
00436
00437 }
        // namespace aerobus
00438
00439 // type list
00440 namespace aerobus {
00442
         template <typename... Ts>
00443
         struct type_list;
00444
00445
          namespace internal {
00446
              template <typename T, typename... Us>
00447
              struct pop_front_h {
00448
                  using tail = type_list<Us...>;
00449
                  using head = T;
00450
00451
00452
              template <size_t index, typename L1, typename L2>
              struct split_h {
00453
00454
              private:
00455
                  static_assert(index <= L2::length, "index ouf of bounds");</pre>
00456
                  using a = typename L2::pop_front::type;
                  using b = typename L2::pop_front::tail;
00457
00458
                  using c = typename L1::template push_back<a>;
00459
00460
               public:
```

```
using head = typename split_h<index - 1, c, b>::head;
using tail = typename split_h<index - 1, c, b>::tail;
00462
00463
                };
00464
00465
                template <typename L1, typename L2>
struct split_h<0, L1, L2> {
00466
                    using head = L1;
00467
00468
                     using tail = L2;
00469
                } ;
00470
00471
                template <size_t index, typename L, typename T>
00472
                struct insert h {
00473
                    static_assert(index <= L::length, "index ouf of bounds");</pre>
00474
                    using s = typename L::template split<index>;
00475
                    using left = typename s::head;
                    using right = typename s::tail;
00476
                    using 11 = typename left::template push_back<T>;
using type = typename l1::template concat<right>;
00477
00478
00479
00480
00481
                template <size_t index, typename L>
00482
                struct remove_h {
                    using s = typename L::template split<index>;
using left = typename s::head;
using right = typename s::tail;
00483
00484
00485
                    using rr = typename right::pop_front::tail;
00486
00487
                     using type = typename left::template concat<rr>;
00488
           } // namespace internal
00489
00490
00494
           template <typename... Ts>
00495
           struct type_list {
00496
00497
                template <typename T>
00498
                struct concat_h;
00499
00500
                template <typename... Us>
                struct concat_h<type_list<Us...» {
00502
                    using type = type_list<Ts..., Us...>;
00503
00504
            public:
00505
               static constexpr size_t length = sizeof...(Ts);
00507
00508
00511
                template <typename T>
00512
                using push_front = type_list<T, Ts...>;
00513
00516
                template <size_t index>
00517
                using at = internal::type_at_t<index, Ts...>;
00518
                struct pop_front {
                    using type = typename internal::pop_front_h<Ts...>::head;
using tail = typename internal::pop_front_h<Ts...>::tail;
00522
00524
00525
                };
00526
00529
                template <typename T>
00530
                using push_back = type_list<Ts..., T>;
00531
00534
                template <typename U>
00535
                using concat = typename concat_h<U>::type;
00536
00539
                template <size_t index>
00540
                struct split {
00541
00542
                    using inner = internal::split_h<index, type_list<>, type_list<Ts...»;</pre>
00543
00544
00545
                    using head = typename inner::head;
                    using tail = typename inner::tail;
00546
00547
00548
00552
                template <typename T, size_t index>
00553
                using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00554
                template <size_t index>
using remove = typename internal::remove_h<index, type_list<Ts...»::type;</pre>
00557
00558
00559
00560
00562
           template <>
00563
           struct type_list<> {
00564
                static constexpr size t length = 0;
00565
00566
                template <typename T>
00567
                using push_front = type_list<T>;
00568
                template <typename T>
00569
                using push_back = type_list<T>;
00570
```

```
00572
              template <typename U>
00573
              using concat = U;
00574
00575
              // TODO(jewave): assert index == 0
              template <typename T, size_t index>
using insert = type_list<T>;
00576
00577
00578
00579 } // namespace aerobus
00580
00581 // i32
00582 namespace aerobus {
00584
          struct i32 {
00585
              using inner_type = int32_t;
00588
              template<int32_t x>
00589
              struct val {
                  using enclosing_type = i32;
00591
                  static constexpr int32_t v = x;
00593
00594
00597
                  template<typename valueType>
00598
                  static constexpr INLINED DEVICE valueType get() { return static_cast<valueType>(x); }
00599
00601
                  using is_zero_t = std::bool_constant<x == 0>;
00602
00604
                  static std::string to_string() {
00605
                     return std::to_string(x);
00606
00607
00610
                  template<typename valueRing>
                  static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
00611
00612
                      return static cast<valueRing>(x);
00613
00614
00615
              using zero = val<0>;
using one = val<1>;
00617
00619
              static constexpr bool is_field = false;
00621
00623
              static constexpr bool is_euclidean_domain = true;
00627
              template<auto x>
00628
              using inject_constant_t = val<static_cast<int32_t>(x)>;
00629
00630
              template<typename v>
00631
              using inject_ring_t = v;
00632
           private:
00633
00634
              template<typename v1, typename v2>
00635
              struct add {
                  using type = val<v1::v + v2::v>;
00636
00637
00638
00639
              template<typename v1, typename v2>
00640
00641
                  using type = val<v1::v - v2::v>;
00642
00643
              template<typename v1, typename v2>
00644
              struct mul {
00646
                  using type = val<v1::v* v2::v>;
00647
00648
00649
              template<typename v1, typename v2>
00650
              struct div {
00651
                  using type = val<v1::v / v2::v>;
00652
00653
00654
              template<typename v1, typename v2>
00655
              struct remainder {
                  using type = val<v1::v % v2::v>;
00656
00657
00659
              template<typename v1, typename v2>
00660
00661
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00662
00663
00664
              template<typename v1, typename v2>
00665
00666
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00667
              };
00668
00669
              template<typename v1, typename v2>
00670
              struct eq {
00671
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00672
00673
00674
              template<typename v1>
00675
              struct pos {
```

```
using type = std::bool_constant<(v1::v > 0)>;
00677
              };
00678
00679
           public:
00685
              template<typename v1, typename v2>
00686
              using add_t = typename add<v1, v2>::type;
00693
              template<typename v1, typename v2>
00694
              using sub_t = typename sub<v1, v2>::type;
00695
00701
              template<typename v1, typename v2>
00702
              using mul_t = typename mul<v1, v2>::type;
00703
00709
              template<typename v1, typename v2>
00710
              using div_t = typename div<v1, v2>::type;
00711
00717
              template<typename v1, typename v2>
00718
              using mod_t = typename remainder<v1, v2>::type;
00725
              template<typename v1, typename v2>
00726
              using gt_t = typename gt<v1, v2>::type;
00727
00733
              template<typename v1, typename v2>
00734
              using lt_t = typename lt<v1, v2>::type;
00735
00741
              template<typename v1, typename v2>
00742
              using eq_t = typename eq<v1, v2>::type;
00743
00748
              template<typename v1, typename v2>
              static constexpr bool eq_v = eq_t<v1, v2>::value;
00749
00750
              template<typename v1, typename v2>
using gcd_t = gcd_t<i32, v1, v2>;
00756
00757
00758
00763
              {\tt template}{<}{\tt typename}\ {\tt v}{>}
00764
              using pos_t = typename pos<v>::type;
00765
00770
              template<typename v>
00771
              static constexpr bool pos_v = pos_t<v>::value;
00772
00773 } // namespace aerobus
00774
00775 // i64
00776 namespace aerobus {
00778
         struct i64 {
00780
             using inner_type = int64_t;
00783
              template<int64_t x>
00784
              struct val {
00786
                  using inner_type = int32_t;
                  using enclosing_type = i64;
00788
00790
                  static constexpr int64_t v = x;
00791
00794
                  template<typename valueType>
                  static constexpr DEVICE INLINED valueType get() {
00795
00796
                       return static_cast<valueType>(x);
00797
                  }
00798
00800
                  using is_zero_t = std::bool_constant<x == 0>;
00801
00803
                  static std::string to_string() {
00804
                       return std::to_string(x);
00805
                  }
00806
00809
                  template<typename valueRing>
00810
                   static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
00811
                       return static_cast<valueRing>(x);
00812
00813
              };
00814
00818
              template<auto x>
00819
              using inject_constant_t = val<static_cast<int64_t>(x)>;
00820
00825
              template<typename v>
00826
              using inject_ring_t = v;
00827
00829
              using zero = val<0>;
00831
              using one = val<1>;
00833
              static constexpr bool is_field = false;
00835
              static constexpr bool is_euclidean_domain = true;
00836
00837
           private:
00838
              template<typename v1, typename v2>
00839
              struct add {
00840
                  using type = val<v1::v + v2::v>;
00841
00842
00843
              template<tvpename v1, tvpename v2>
```

```
00844
              struct sub {
                  using type = val<v1::v - v2::v>;
00845
00846
              };
00847
              template<typename v1, typename v2> ^{\circ}
00848
00849
              struct mul {
00850
                  using type = val<v1::v* v2::v>;
00851
00852
00853
              template<typename v1, typename v2>
00854
              struct div {
                 using type = val<v1::v / v2::v>;
00855
00856
00857
00858
              template<typename v1, typename v2>
00859
              struct remainder {
                  using type = val<v1::v% v2::v>;
00860
00861
00862
00863
              template<typename v1, typename v2>
00864
00865
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00866
00867
00868
              template<typename v1, typename v2>
00869
              struct lt {
00870
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00871
00872
00873
              template<typename v1, typename v2>
00874
              struct eq {
00875
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00876
00877
00878
              template<typename v>
00879
              struct pos {
00880
                  using type = std::bool_constant<(v::v > 0)>;
00882
00883
           public:
00888
              template<typename v1, typename v2> \,
00889
              using add_t = typename add<v1, v2>::type;
00890
00895
              template<typename v1, typename v2>
00896
              using sub_t = typename sub<v1, v2>::type;
00897
00902
              template<typename v1, typename v2>
00903
              using mul_t = typename mul<v1, v2>::type;
00904
00910
              template<typename v1, typename v2>
00911
              using div_t = typename div<v1, v2>::type;
00912
00917
              template<typename v1, typename v2>
00918
              using mod_t = typename remainder<v1, v2>::type;
00919
00925
              template<typename v1, typename v2>
              using gt_t = typename gt<v1, v2>::type;
00926
00927
00932
              template<typename v1, typename v2>
00933
              static constexpr bool gt_v = gt_t<v1, v2>::value;
00934
00940
              template<typename v1, typename v2>
00941
              using lt_t = typename lt<v1, v2>::type;
00942
00948
              template<typename v1, typename v2>
00949
              static constexpr bool lt_v = lt_t<v1, v2>::value;
00950
00956
              template<typename v1, typename v2>
00957
              using eq_t = typename eq<v1, v2>::type;
00958
00964
              template<typename v1, typename v2>
00965
              static constexpr bool eq_v = eq_t<v1, v2>::value;
00966
00972
              template<typename v1, typename v2> ^{\circ}
00973
              using gcd_t = gcd_t < i64, v1, v2>;
00974
00979
              template<typename v>
00980
              using pos_t = typename pos<v>::type;
00981
00986
              template<tvpename v>
00987
              static constexpr bool pos_v = pos_t<v>::value;
00988
          };
00989
00991
          template<>
00992
          struct Embed<i32, i64> {
00995
              template<typename val>
00996
              using type = i64::val<static cast<int64 t>(val::v)>;
```

```
};
00998 } // namespace aerobus
00999
01000 // z/pz
01001 namespace aerobus {
         template<int32_t p>
01007
01008
         struct zpz {
01010
             using inner_type = int32_t;
01011
01014
              template<int32_t x>
01015
              struct val {
                 using enclosing_type = zpz;
01017
01019
                  static constexpr int32_t v = x % p;
01020
01023
                  template<typename valueType>
01024
                  static constexpr DEVICE INLINED valueType get() { return static_cast<valueType>(x % p); }
01025
01027
                  using is zero t = std::bool constant<v == 0>;
01028
01030
                  static constexpr bool is_zero_v = v == 0;
01031
01034
                  static std::string to_string() {
01035
                     return std::to_string(x % p);
01036
01037
01038
                  template<typename valueRing>
                  static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
01039
01040
                     return static_cast<valueRing>(x % p);
01041
01042
              };
01043
01046
              template<auto x>
01047
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01048
01050
              using zero = val<0>;
01051
01053
             using one = val<1>;
01054
01056
              static constexpr bool is_field = is_prime::value;
01057
01059
              static constexpr bool is_euclidean_domain = true;
01060
01061
           private:
01062
              template<typename v1, typename v2>
01063
              struct add {
01064
                 using type = val<(v1::v + v2::v) % p>;
01065
01066
              template<typename v1, typename v2>
01067
01068
              struct sub {
01069
                 using type = val<(v1::v - v2::v) % p>;
01070
01071
01072
              template<typename v1, typename v2>
01073
              struct mul {
01074
                 using type = val<(v1::v* v2::v) % p>;
01075
01076
01077
              template<typename v1, typename v2>
01078
              struct div {
                 using type = val<(v1::v% p) / (v2::v % p)>;
01079
01080
              };
01081
01082
              template<typename v1, typename v2>
01083
              struct remainder {
01084
                 using type = val<(v1::v% v2::v) % p>;
01085
01086
01087
              template<typename v1, typename v2>
01088
              struct gt {
01089
                 using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01090
01091
01092
              template<typename v1, typename v2> ^{\circ}
01093
              struct lt {
01094
                  using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01095
01096
01097
              template<typename v1, typename v2>
01098
              struct eq {
                 using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01099
01100
01101
01102
              template<typename v1>
01103
              struct pos {
                  using type = std::bool_constant<(v1::v > 0)>;
01104
01105
              };
```

```
01106
01107
           public:
01111
              template<typename v1, typename v2>
01112
              using add_t = typename add<v1, v2>::type;
01113
01117
              template<typename v1, typename v2>
01118
              using sub_t = typename sub<v1, v2>::type;
01119
01123
              template<typename v1, typename v2>
01124
              using mul_t = typename mul<v1, v2>::type;
01125
01129
              template<typename v1, typename v2>
01130
              using div_t = typename div<v1, v2>::type;
01131
01135
              template<typename v1, typename v2>
01136
              using mod_t = typename remainder<v1, v2>::type;
01137
              template<typename v1, typename v2>
using gt_t = typename gt<v1, v2>::type;
01141
01142
01143
01147
              template<typename v1, typename v2>
01148
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01149
              template<typename v1, typename v2>
using lt_t = typename lt<v1, v2>::type;
01153
01154
01155
01159
              template<typename v1, typename v2>
01160
              static constexpr bool lt_v = lt_t<v1, v2>::value;
01161
01165
              template<typename v1, typename v2>
01166
              using eq_t = typename eq<v1, v2>::type;
01167
01171
              template<typename v1, typename v2>
01172
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01173
01177
              template<typename v1, typename v2>
              using gcd_t = gcd_t<i32, v1, v2>;
01178
01179
01182
              template<typename v1>
01183
              using pos_t = typename pos<v1>::type;
01184
01187
              template<typename v>
01188
              static constexpr bool pos v = pos t<v>::value;
01189
          };
01190
01193
          template<int32_t x>
01194
          struct Embed<zpz<x>, i32> {
01197
              template <typename val>
01198
              using type = i32::val<val::v>;
01199
01200 } // namespace aerobus
01201
01202 // polynomial
01203 namespace aerobus {
          // coeffN x^N + ...
01204
01209
          template<typename Ring>
01210
          requires IsEuclideanDomain<Ring>
01211
01212
             static constexpr bool is_field = false;
01213
              static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01214
01218
              template<typename coeffN, typename... coeffs>
01219
              struct val {
01221
                 using ring_type = Ring;
01223
                  using enclosing_type = polynomial<Ring>;
01225
                  static constexpr size_t degree = sizeof...(coeffs);
01227
                  using aN = coeffN;
01229
                  using strip = val<coeffs...>;
01231
                  using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01233
                  static constexpr bool is_zero_v = is_zero_t::value;
01234
               private:
01235
01236
                  template<size_t index, typename E = void>
01237
                  struct coeff at {}:
01238
01239
                   template<size_t index>
01240
                  struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))» {</pre>
01241
                      using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01242
                  };
01243
01244
                  template<size t index>
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))» {
01246
                      using type = typename Ring::zero;
01247
                  };
01248
01249
               public:
01252
                  template<size t index>
```

```
using coeff_at_t = typename coeff_at<index>::type;
01254
01257
                   static std::string to_string() {
01258
                       return string_helper<coeffN, coeffs...>::func();
01259
01260
01265
                   template<typename valueRing>
01266
                   static constexpr DEVICE INLINED valueRing eval(const valueRing& x) {
                     return horner_evaluation<valueRing, val>
    ::template inner<0, degree + 1>
01267
01268
                                ::func(static_cast<valueRing>(0), x);
01269
01270
                   }
01271
               };
01272
01275
               template<typename coeffN>
               struct val<coeffN> {
    using ring_type = Ring;
01276
01278
                   using ring_type = king;
using enclosing_type = polynomial<Ring>;
static constexpr size_t degree = 0;
01280
01282
                   using aN = coeffN;
01283
01284
                   using strip = val<coeffN>;
01285
                   using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01286
01287
                   static constexpr bool is zero v = is zero t::value;
01288
                   template<size_t index, typename E = void>
01289
01290
                   struct coeff_at {};
01291
01292
                   template<size_t index>
                   struct coeff_at<index, std::enable_if_t<(index == 0)» {</pre>
01293
01294
                       using type = aN;
01295
01296
01297
                   template<size_t index>
01298
                   struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01299
                       using type = typename Ring::zero;
01300
                   };
01301
01302
                   template<size_t index>
01303
                   using coeff_at_t = typename coeff_at<index>::type;
01304
01305
                   static std::string to_string() {
                      return string_helper<coeffN>::func();
01306
01307
01308
01309
                   template<typename valueRing>
01310
                   static constexpr DEVICE INLINED valueRing eval(const valueRing& x) {
01311
                       return static_cast<valueRing>(aN::template get<valueRing>());
01312
01313
              };
01314
01316
               using zero = val<typename Ring::zero>;
01318
               using one = val<typename Ring::one>;
01320
              using X = val<typename Ring::one, typename Ring::zero>;
01321
01322
           private:
              template<typename P, typename E = void>
01324
               struct simplify;
01325
01326
               template <typename P1, typename P2, typename I>
01327
               struct add low;
01328
01329
               template<typename P1, typename P2>
01330
               struct add {
01331
                   using type = typename simplify<typename add_low<
                  Р1,
01332
01333
                  P2.
                   internal::make_index_sequence_reverse<</pre>
01334
01335
                   std::max(P1::degree, P2::degree) + 1
01336
                   »::type>::type;
01337
01338
01339
               template <typename P1, typename P2, typename I>
01340
               struct sub low:
01341
01342
               template <typename P1, typename P2, typename I>
01343
               struct mul_low;
01344
01345
               template<typename v1, typename v2>
01346
               struct mul {
01347
                       using type = typename mul_low<
01348
                           v1,
01349
01350
                           internal::make_index_sequence_reverse<
01351
                           v1::degree + v2::degree + 1
01352
                           »::type;
01353
               };
```

```
01355
               template<typename coeff, size_t deg>
01356
               struct monomial;
01357
01358
               template<typename v, typename E = void>
01359
               struct derive helper {};
01360
01361
               template<typename v>
01362
               struct derive_helper<v, std::enable_if_t<v::degree == 0» {</pre>
01363
                   using type = zero;
01364
               };
01365
01366
               template<typename v>
01367
               struct derive_helper<v, std::enable_if_t<v::degree != 0» {
01368
                   using type = typename add<
01369
                       typename derive_helper<typename simplify<typename v::strip>::type>::type,
                       typename monomial<
01370
01371
                           typename Ring::template mul_t<
01372
                                typename v::aN,
01373
                                typename Ring::template inject_constant_t<(v::degree)>
01374
01375
                           v::degree - 1
01376
                       >::type
01377
                   >::type;
01378
               };
01379
01380
               template<typename v1, typename v2, typename E = void>
01381
               struct eq_helper {};
01382
              template<typename v1, typename v2>
struct eg_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {</pre>
01383
01384
01385
                  using type = std::false_type;
01386
01387
01388
               template<typename v1, typename v2>
struct eq_helper<v1, v2, std::enable_if_t<
    v1::degree == v2::degree &&</pre>
01389
01390
01391
01392
                   (v1::degree != 0 || v2::degree != 0) &&
01393
                   std::is_same<
01394
                   typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01395
                   std::false_type
01396
                   >::value
01397
01398
               > {
01399
                   using type = std::false_type;
01400
01401
01402
               template<typename v1, typename v2>
               struct eq_helper<v1, v2, std::enable_if_t<
01403
                   v1::degree == v2::degree &&
01404
01405
                   (v1::degree != 0 || v2::degree != 0) &&
                   std::is_same<
01406
01407
                   typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01408
                   std::true_type
01409
                   >::value
01410
               » {
01411
                   using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01412
01413
01414
               template<typename v1, typename v2>
               struct eq_helper<v1, v2, std::enable_if_t<
01415
01416
                   v1::degree == v2::degree &&
01417
                   (v1::degree == 0)
01418
01419
                   using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01420
              };
01421
01422
               template<typename v1, typename v2, typename E = void>
01423
               struct lt_helper {};
01424
01425
               template<typename v1, typename v2>
01426
               struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {</pre>
01427
                  using type = std::true_type;
01428
01429
01430
               template<typename v1, typename v2>
01431
               struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01432
                   using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01433
01434
01435
               template<typename v1, typename v2>
01436
               struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
01437
                   using type = std::false_type;
01438
01439
01440
               template<typename v1, typename v2, typename E = void>
```

```
01441
              struct gt_helper {};
01442
01443
              template<typename v1, typename v2>
              01444
01445
                  using type = std::true_type;
01446
01447
01448
              template<typename v1, typename v2> ^{\circ}
01449
              struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01450
                  using type = std::false_type;
01451
01452
01453
              template<typename v1, typename v2>
01454
              struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {</pre>
01455
                 using type = std::false_type;
01456
01457
01458
              // when high power is zero : strip
01459
              template<typename P>
              struct simplify<P, std::enable_if_t<
01460
01461
                 std::is_same<
01462
                  typename Ring::zero,
01463
                  typename P::aN
01464
                  >::value && (P::degree > 0)
01465
              » {
01466
                  using type = typename simplify<typename P::strip>::type;
01467
              } ;
01468
              // otherwise : do nothing
01469
01470
              template<typename P>
              struct simplify<P, std::enable_if_t<
01471
01472
                  !std::is_same<
01473
                  typename Ring::zero,
01474
                  typename P::aN
01475
                 >::value && (P::degree > 0)
              » {
01476
01477
                  using type = P;
01478
              };
01479
01480
              // do not simplify constants
01481
              template<typename P>
              struct simplify<P, std::enable_if_t<P::degree == 0» {</pre>
01482
01483
                 using type = P;
01484
01485
              // addition at
01486
01487
              template<typename P1, typename P2, size_t index>
01488
              struct add at {
01489
                  using type =
01490
                      typename Ring::template add_t<</pre>
01491
                          typename P1::template coeff_at_t<index>,
01492
                          typename P2::template coeff_at_t<index>>;
01493
01494
              template<typename P1, typename P2, size_t index>
01495
01496
              using add_at_t = typename add_at<P1, P2, index>::type;
01498
              template<typename P1, typename P2, std::size_t... I>
01499
              struct add_low<P1, P2, std::index_sequence<I...» {
01500
                  using type = val<add_at_t<P1, P2, I>...>;
01501
01502
01503
              // substraction at
01504
              template<typename P1, typename P2, size_t index>
01505
              struct sub_at {
01506
                  using type =
01507
                      typename Ring::template sub_t<</pre>
                          typename P1::template coeff_at_t<index>,
01508
01509
                          typename P2::template coeff_at_t<index>>;
              };
01511
01512
              template<typename P1, typename P2, size_t index>
01513
              using sub_at_t = typename sub_at<P1, P2, index>::type;
01514
              template<typename P1, typename P2, std::size_t... I>
struct sub_low<P1, P2, std::index_sequence<I...» {</pre>
01515
01516
01517
                 using type = val<sub_at_t<P1, P2, I>...>;
01518
01519
01520
              template<typename P1, typename P2>
01521
              struct sub {
01522
                  using type = typename simplify<typename sub_low<
01523
01524
                  P2,
01525
                  internal::make_index_sequence_reverse<</pre>
01526
                  std::max(P1::degree, P2::degree) + 1
01527
                  »::type>::type;
```

```
01528
               };
01529
01530
               // multiplication at
01531
               template<typename v1, typename v2, size_t k, size_t index, size_t stop>
01532
               struct mul_at_loop_helper {
                   using type = typename Ring::template add_t<
01533
                        typename Ring::template mul_t<
01534
01535
                        typename v1::template coeff_at_t<index>,
01536
                        typename v2::template coeff_at_t<k - index>
01537
01538
                        typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
01539
01540
               };
01541
01542
               template<typename v1, typename v2, size_t k, size_t stop>
01543
               struct mul_at_loop_helper<v1, v2, k, stop, stop> {
                   using type = typename Ring::template mul_t<
   typename v1::template coeff_at_t<stop>,
   typename v2::template coeff_at_t<0>>;
01544
01545
01546
01547
               };
01548
01549
               template <typename v1, typename v2, size_t k, typename E = void>
01550
               struct mul_at {};
01551
01552
               template<typename v1, typename v2, size_t k>
               struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)» {
01553
01554
                   using type = typename Ring::zero;
01555
01556
01557
               template<typename v1, typename v2, size_t k> struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)» {
01558
01559
                  using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01560
01561
01562
               template<typename P1, typename P2, size_t index>
01563
               using mul_at_t = typename mul_at<P1, P2, index>::type;
01564
01565
               template<typename P1, typename P2, std::size_t... I>
01566
               struct mul_low<P1, P2, std::index_sequence<I...» {
01567
                  using type = val<mul_at_t<P1, P2, I>...>;
01568
01569
01570
               // division helper
01571
               template< typename A, typename B, typename Q, typename R, typename E = void>
01572
               struct div helper {};
01573
01574
               template<typename A, typename B, typename Q, typename R>
               struct div_helper<A, B, Q, R, std::enable_if_t<
    (R::degree < B::degree) ||</pre>
01575
01576
                   (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
01577
01578
                   using q_type = Q;
01579
                   using mod_type = R;
01580
                   using gcd_type = B;
01581
               };
01582
               template<typename A, typename B, typename Q, typename R>
01583
               struct div_helper<A, B, Q, R, std::enable_if_t<
                    (R::degree >= B::degree) &&
01585
01586
                   !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
                private: // NOLINT
   using rN = typename R::aN;
01587
01588
                   using bN = typename B::aN;
01589
01590
                   using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
     B::degree>::type;
01591
                   using rr = typename sub<R, typename mul<pT, B>::type>::type;
01592
                   using qq = typename add<Q, pT>::type;
01593
01594
                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;
01595
01596
01597
                   using gcd_type = rr;
01598
               };
01599
01600
               template<typename A, typename B>
01601
               struct div {
                   static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
01602
01603
                   using q_type = typename div_helper<A, B, zero, A>::q_type;
01604
                   using m_type = typename div_helper<A, B, zero, A>::mod_type;
01605
               };
01606
01607
               template<typename P>
01608
               struct make_unit {
01609
                   using type = typename div<P, val<typename P::aN>>::q_type;
01610
01611
01612
               template<typename coeff, size_t deg>
01613
               struct monomial {
```

```
using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
01615
01616
01617
              template<typename coeff>
01618
              struct monomial<coeff, 0> {
                  using type = val<coeff>;
01619
01620
01621
01622
              template<typename valueRing, typename P>
01623
              struct horner evaluation {
01624
                  template<size_t index, size_t stop>
01625
                  struct inner {
                      static constexpr DEVICE INLINED valueRing func (const valueRing& accum, const
01626
      valueRing& x) {
01627
                           constexpr valueRing coeff =
01628
                              static_cast<valueRing>(P::template coeff_at_t<P::degree - index>::template
get<valueRing>());
01629
                           return horner evaluation<valueRing, P>::template inner<index + 1, stop>::func(x *
     accum + coeff, x);
01630
01631
01632
01633
                  template<size_t stop>
01634
                  struct inner<stop, stop> {
                      static constexpr DEVICE INLINED valueRing func (const valueRing& accum, const
01635
      valueRing& x) {
01636
                           return accum;
01637
01638
                  };
01639
              };
01640
01641
              template<typename coeff, typename... coeffs>
01642
              struct string_helper {
01643
                  static std::string func() {
                       std::string tail = string_helper<coeffs...>::func();
std::string result = "";
01644
01645
01646
                       if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
01647
                           return tail;
01648
                       } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
                          if (sizeof...(coeffs) == 1) {
    result += "x";
01649
01650
                           } else {
01651
                               result += "x^" + std::to string(sizeof...(coeffs));
01652
01653
                           }
01654
                       } else {
01655
                           if (sizeof...(coeffs) == 1) {
01656
                               result += coeff::to_string() + " x";
01657
                           } else {
                               result += coeff::to string()
01658
                                        + " x^" + std::to_string(sizeof...(coeffs));
01659
01660
                           }
01661
                       }
01662
                       if (!tail.empty()) {
    result += " + " + tail;
01663
01664
01665
01666
01667
                       return result:
01668
01669
              };
01670
01671
              template<typename coeff>
01672
              struct string_helper<coeff>
01673
                  static std::string func() {
01674
                       if (!std::is_same<coeff, typename Ring::zero>::value) {
01675
                           return coeff::to_string();
01676
                       } else {
                           return "";
01677
01678
01679
                  }
01680
              };
01681
           public:
01682
              template<typename P>
01685
01686
              using simplify t = typename simplify<P>::type;
01687
01691
               template<typename v1, typename v2>
01692
              using add_t = typename add<v1, v2>::type;
01693
01697
              template<typename v1, typename v2>
01698
              using sub t = typename sub<v1, v2>::type;
01699
01703
               template<typename v1, typename v2>
01704
              using mul_t = typename mul<v1, v2>::type;
01705
01709
              template<typename v1, typename v2>
01710
              using eq_t = typename eq_helper<v1, v2>::type;
```

```
01711
              template<typename v1, typename v2>
01715
01716
              using lt_t = typename lt_helper<v1, v2>::type;
01717
01721
              template<typename v1, typename v2> ^{\circ}
01722
              using gt_t = typename gt_helper<v1, v2>::type;
01723
01727
              template<typename v1, typename v2>
01728
              using div_t = typename div<v1, v2>::q_type;
01729
01733
              template<typename v1, typename v2>
01734
              using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
01735
01739
              template<typename coeff, size_t deg>
01740
              using monomial_t = typename monomial<coeff, deg>::type;
01741
01744
              template<typename v>
              using derive_t = typename derive_helper<v>::type;
01745
01749
              template<typename v>
01750
              using pos_t = typename Ring::template pos_t<typename v::aN>;
01751
01754
              template<typename v>
              static constexpr bool pos_v = pos_t<v>::value;
01755
01756
01760
              template<typename v1, typename v2>
01761
              using gcd_t = std::conditional_t<
01762
                  Ring::is_euclidean_domain,
01763
                  typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
01764
                  void>;
01765
01769
              template<auto x>
01770
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
01771
01775
              template < typename v >
01776
              using inject_ring_t = val<v>;
01777
          };
01778 } // namespace aerobus
01779
01780 // fraction field
01781 namespace aerobus {
01782
         namespace internal {
             template<typename Ring, typename E = void>
01783
01784
              requires IsEuclideanDomain<Ring>
01785
              struct _FractionField {};
01786
01787
              template<typename Ring>
              requires IsEuclideanDomain<Ring>
01788
              struct _FractionFieldRing, std::enable_if_t<Ring::is_euclidean_domain<pre>% static constexpr bool is_field = true;
01789
01791
01792
                  static constexpr bool is_euclidean_domain = true;
01793
01794
               private:
01795
                  template<typename val1, typename val2, typename E = void>
01796
                  struct to_string_helper {};
01797
01798
                  template<typename val1, typename val2>
01799
                  struct to_string_helper <val1, val2,
01800
                       std::enable_if_t<
01801
                       Ring::template eq_t<
                       val2, typename Ring::one
01802
01803
                      >::value
01804
01805
                  > {
01806
                       static std::string func() {
01807
                          return vall::to_string();
01808
01809
                  };
01810
                  template<typename val1, typename val2>
01812
                  struct to_string_helper<val1, val2,
01813
                       std::enable_if_t<
01814
                       !Ring::template eq_t<
01815
                       val2.
01816
                       typename Ring::one
01817
                       >::value
01818
01819
01820
                       static std::string func() {
                           return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
01821
01822
01823
                  };
01824
01825
               public:
01829
                  template<typename val1, typename val2>
01830
                  struct val {
01832
                      using x = val1;
```

```
using y = val2;
                      using is_zero_t = typename vall::is_zero_t;
01836
01838
                      static constexpr bool is_zero_v = val1::is_zero_t::value;
01839
01841
                      using ring_type = Ring;
                      using enclosing_type = _FractionField<Ring>;
01842
01843
01846
                       static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
01847
01851
                      template<typename valueType>
                      static constexpr DEVICE INLINED valueType get() {
01852
                          return static_cast<valueType>(x::v) / static_cast<valueType>(y::v);
01853
01854
01855
01858
                      static std::string to_string() {
01859
                          return to_string_helper<val1, val2>::func();
01860
01861
01866
                      template<typename valueRing>
                      static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
01867
01868
                          return x::eval(v) / y::eval(v);
01869
01870
                  };
01871
01873
                  using zero = val<typename Ring::zero, typename Ring::one>;
                  using one = val<typename Ring::one, typename Ring::one>;
01875
01876
01879
                  template<typename v>
01880
                  using inject_t = val<v, typename Ring::one>;
01881
01884
                  template<auto x>
01885
                  using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
      Ring::one>;
01886
01889
                  template < typename v >
                  using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
01890
01891
                  using ring_type = Ring;
01893
01894
01895
               private:
01896
                  template<typename v, typename E = void>
01897
                  struct simplify {};
01898
01899
01900
                  template<typename v>
01901
                  struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {</pre>
01902
                      using type = typename _FractionField<Ring>::zero;
01903
                  };
01904
01905
                  // x != 0
01906
                  template<typename v>
01907
                  struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value» {
                   private:
01908
01909
                      using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
                      using newx = typename Ring::template div_t<typename v::x, _gcd>;
01910
                      using newy = typename Ring::template div_t<typename v::y, _gcd>;
01911
01912
01913
                      using posx = std::conditional t<
01914
                                           !Ring::template pos_v<newy>,
01915
                                           typename Ring::template sub_t<typename Ring::zero, newx>,
01916
                                          newx>:
01917
                      using posy = std::conditional_t<
01918
                                           !Ring::template pos_v<newy>,
01919
                                           typename Ring::template sub_t<typename Ring::zero, newy>,
01920
                   public:
01921
01922
                      using type = typename _FractionField<Ring>::template val<posx, posy>;
01923
                  };
01924
01925
               public:
01928
                 template<typename v>
01929
                  using simplify_t = typename simplify<v>::type;
01930
01931
                  template<typename v1, typename v2>
01932
01933
                  struct add {
01934
01935
                      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>;
01936
01937
                      using dividend = typename Ring::template add t<a, b>;
                      using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01938
01939
                      using g = typename Ring::template gcd_t<dividend, diviser>;
01940
                   public:
01941
01942
                      using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
     diviser»:
01943
                  };
```

```
01945
                                 template<typename v>
01946
                                  struct pos
01947
                                        using type = std::conditional_t<
                                                 \label{eq:condition} $$(\text{Ring::template pos_v<typename v::y>}) \mid | $$
01948
                                                 (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
01949
01950
                                                std::true_type,
01951
                                                std::false_type>;
01952
                                 } ;
01953
01954
                                 template<typename v1, typename v2>
01955
                                  struct sub {
01956
                                   private:
01957
                                         using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01958
                                         using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
                                         using dividend = typename Ring::template sub_t<a, b>;
using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01959
01960
01961
                                         using g = typename Ring::template gcd_t<dividend, diviser>;
01962
01963
                                   public:
                                         using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
01964
          diviser»;
01965
                                  };
01966
01967
                                 template<typename v1, typename v2>
01968
                                  struct mul {
01969
01970
                                         using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
01971
                                         using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01972
01973
                                   public:
01974
                                        using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
01975
01976
01977
                                  template<typename v1, typename v2, typename E = void>
01978
                                  struct div {}:
01979
01980
                                  template<typename v1, typename v2>
01981
                                  struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
           _FractionField<Ring>::zero>::value» {
01982
01983
                                         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>;
01984
01985
01986
                                         using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
01987
01988
01989
01990
                                  template<typename v1, typename v2>
01991
                                  struct div<v1, v2, std::enable if t<
01992
                                         std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
01993
                                         using type = one;
01994
01995
01996
                                  template<typename v1, typename v2>
01997
                                 struct eq {
01998
                                        using type = std::conditional_t<
01999
                                                        std::is_same<typename simplify_t<vl>::x, typename simplify_t<v2>::x>::value &&
02000
                                                        std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
                                                std::true_type,
02001
02002
                                                std::false_type>;
02003
                                 };
02004
02005
                                  template<typename v1, typename v2, typename E = void>
02006
                                  struct gt;
02007
02008
                                 template<typename v1, typename v2>
struct qt<v1, v2, std::enable_if_t<</pre>
02009
                                         (eq<v1, v2>::type::value)
02010
02011
02012
                                         using type = std::false_type;
02013
02014
                                  template<typename v1, typename v2>
02015
                                 struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02016
02017
02018
                                         (!pos<v1>::type::value) && (!pos<v2>::type::value)
02019
02020
                                         using type = typename gt<
                                                typename sub<zero, v1>::type, typename sub<zero, v2>::type
02021
02022
                                         >::type;
02023
                                 } ;
02024
02025
                                  template<typename v1, typename v2>
02026
                                  struct gt<v1, v2, std::enable_if_t<
                                         (!eq<v1, v2>::type::value) &&
02027
02028
                                          (pos<v1>::type::value) && (!pos<v2>::type::value)
```

```
02029
02030
                       using type = std::true_type;
02031
                  };
02032
02033
                  template<typename v1, typename v2>
                  struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02034
02035
02036
                       (!pos<v1>::type::value) && (pos<v2>::type::value)
02037
02038
                       using type = std::false_type;
02039
                  };
02040
02041
                  template<typename v1, typename v2>
02042
                  struct gt<v1, v2, std::enable_if_t<
02043
                       (!eq<v1, v2>::type::value) &&
02044
                       (pos<v1>::type::value) && (pos<v2>::type::value)
02045
02046
                       using type = typename Ring::template gt t<
                          typename Ring::template mul_t<v1::x, v2::y>,
02048
                           typename Ring::template mul_t<v2::y, v2::x>
02049
02050
                  };
02051
02052
               public:
02057
                  template<typename v1, typename v2>
02058
                  using add_t = typename add<v1, v2>::type;
02059
02064
                  template<typename v1, typename v2>
02065
                  using mod t = zero;
02066
02071
                  template<typename v1, typename v2>
02072
                  using gcd_t = v1;
02073
02077
                  template<typename v1, typename v2>
02078
                  using sub_t = typename sub<v1, v2>::type;
02079
02083
                  template<typename v1, typename v2>
02084
                  using mul_t = typename mul<v1, v2>::type;
02085
02089
                  template<typename v1, typename v2>
02090
                  using div_t = typename div<v1, v2>::type;
02091
                  template<typename v1, typename v2>
02095
02096
                  using eq_t = typename eq<v1, v2>::type;
02097
02101
                  template<typename v1, typename v2>
02102
                  static constexpr bool eq_v = eq<v1, v2>::type::value;
02103
02107
                  template<typename v1, typename v2>
                  using gt_t = typename gt<v1, v2>::type;
02108
02113
                  template<typename v1, typename v2>
02114
                  static constexpr bool gt_v = gt<v1, v2>::type::value;
02115
02118
                  template<typename v1>
                  using pos_t = typename pos<v1>::type;
02119
02120
02123
                  template<typename v>
02124
                   static constexpr bool pos_v = pos_t<v>::value;
02125
              };
02126
02127
              template<typename Ring, typename E = void>
02128
              requires IsEuclideanDomain<Ring>
02129
              struct FractionFieldImpl {};
02130
02131
              // fraction field of a field is the field itself
02132
              template<typename Field>
              requires IsEuclideanDomain<Field>
02133
02134
              struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {</pre>
02135
                  using type = Field;
02136
                  template<typename v>
02137
                  using inject_t = v;
02138
              };
02139
              // fraction field of a ring is the actual fraction field
02140
02141
              template<typename Ring>
02142
              requires IsEuclideanDomain<Ring>
02143
              struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field> {
02144
                  using type = _FractionField<Ring>;
02145
              }:
          } // namespace internal
02146
02147
02151
          template<typename Ring>
02152
          requires IsEuclideanDomain<Ring>
02153
          using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02154
02157
          template<typename Ring>
```

```
struct Embed<Ring, FractionField<Ring» {</pre>
02161
              template<typename v>
02162
              using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02163
02164 }
         // namespace aerobus
02165
02166
02167 // short names for common types
02168 namespace aerobus {
02171
          using q32 = FractionField<i32>;
02172
02175
          using fpq32 = FractionField<polynomial<q32>>;
02176
02179
          using q64 = FractionField<i64>;
02180
02182
          using pi64 = polynomial<i64>;
02183
02185
          using pg64 = polynomial<g64>;
02186
02188
          using fpq64 = FractionField<polynomial<q64>>;
02189
02194
          template<typename Ring, typename v1, typename v2>
02195
          using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02196
02203
          template<typename v>
02204
          using embed_int_poly_in_fractions_t =
02205
                   typename Embed<
                       polynomial<typename v::ring_type>,
02206
02207
                       polynomial<FractionField<typename v::ring_type>»::template type<v>;
02208
02212
          template<int64_t p, int64_t q>
using make_q64_t = typename q64::template simplify_t<</pre>
02213
02214
                       typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02215
          template<int32_t p, int32_t q>
using make_q32_t = typename q32::template simplify_t<</pre>
02219
02220
                       typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>»;
02221
02222
02227
          template<typename Ring, typename v1, typename v2>
02228
          using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02233
          template<typename Ring, typename v1, typename v2>
          using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02234
02235
02237
          template<>
02238
          struct Embed<q32, q64> {
              template<typename v>
02241
02242
              using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02243
          };
02244
02248
          template<tvpename Small, tvpename Large>
02249
          struct Embed<polynomial<Small>, polynomial<Large» {</pre>
02250
          private:
02251
              template<typename v, typename i>
02252
              struct at_low;
02253
02254
              template<typename v, size t i>
02255
              struct at_index {
                  using type = typename Embed<Small, Large>::template
02256
      type<typename v::template coeff_at_t<i>>>;
02257
              };
02258
02259
              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...>;
02260
02261
02262
              };
02263
02264
           public:
              template<typename v>
02267
02268
              using type = typename at low<v, typename internal::make index sequence reverse<v::degree +
     1»::type;
02269
02270
02274
          template<typename Ring, auto... xs>
          using make_int_polynomial_t = typename polynomial<Ring>::template val<</pre>
02275
02276
                   typename Ring::template inject_constant_t<xs>...>;
02277
02281
          template<typename Ring, auto... xs>
02282
          using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<
02283
                  typename FractionField<Ring>::template inject_constant_t<xs>...>;
02284 } // namespace aerobus
02285
02286 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02287 namespace aerobus {
02288
          namespace internal {
02289
              template<typename T, size_t x, typename E = void>
02290
              struct factorial {};
02291
```

```
template<typename T, size_t x>
              struct factorial<T, x, std::enable_if_t<(x > 0)» {
02293
              private:
02294
02295
                  template<typename, size_t, typename>
02296
                  friend struct factorial;
02297
              public:
02298
                 using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
     x - 1>::type>;
02299
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02300
              };
02301
02302
              template<typename T>
02303
              struct factorial<T, 0> {
02304
              public:
                using type = typename T::one;
02305
                  static constexpr typename T::inner_type value = type::template get<typename
02306
     T::inner_type>();
02307
             };
02308
          } // namespace internal
02309
02313
          template<typename T, size_t i>
02314
          using factorial_t = typename internal::factorial<T, i>::type;
02315
02319
          template<typename T, size_t i>
          inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02320
02321
          namespace internal {
02322
              template<typename T, size_t k, size_t n, typename E = void>
02323
02324
              struct combination_helper {};
02325
02326
              template<typename T, size t k, size t n>
02327
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
02328
                  using type = typename FractionField<T>::template mul_t<</pre>
02329
                      typename combination_helper<T, k - 1, n - 1>::type,
                      makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02330
02331
              };
02332
02333
              template<typename T, size_t k, size_t n>
02334
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)» {
02335
                  using type = typename combination_helper<T, n - k, n>::type;
02336
              };
02337
02338
              template<typename T, size_t n>
02339
              struct combination_helper<T, 0, n> {
02340
                  using type = typename FractionField<T>::one;
02341
02342
              template<typename T, size_t k, size_t n>
02343
02344
              struct combination {
02345
                  using type = typename internal::combination_helper<T, k, n>::type::x;
02346
                  static constexpr typename T::inner_type value
02347
                              internal::combination_helper<T, k, n>::type::template get<typename</pre>
     T::inner_type>();
02348
              };
02349
          } // namespace internal
02350
02353
          template<typename T, size_t k, size_t n>
02354
          using combination_t = typename internal::combination<T, k, n>::type;
02355
02360
          template<typename T, size_t k, size_t n>
inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02361
02362
02363
          namespace internal {
02364
              template<typename T, size_t m>
02365
              struct bernoulli;
02366
              template<typename T, typename accum, size_t k, size_t m>
02367
02368
              struct bernoulli_helper {
02369
                  using type = typename bernoulli_helper<
02370
02371
                      addfractions_t<T,
02372
                          accum,
02373
                          mulfractions_t<T,
02374
                              makefraction t<T,
02375
                                   combination_t<T, k, m + 1>,
02376
                                   typename T::one>,
02377
                               typename bernoulli<T, k>::type
02378
02379
                      >.
                      k + 1,
02380
02381
                      m>::type;
02382
02383
02384
              template<typename T, typename accum, size_t m>
02385
              struct bernoulli_helper<T, accum, m, m> {
02386
                  using type = accum;
```

```
02387
              };
02388
02389
02390
02391
              template<typename T, size t m>
02392
              struct bernoulli {
02393
                  using type = typename FractionField<T>::template mul_t<</pre>
02394
                      typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02395
                      makefraction_t<T,
02396
                      typename T::template val<static_cast<typename T::inner_type>(-1)>,
02397
                      typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02398
02399
                  >;
02400
02401
                  template<typename floatType>
02402
                  static constexpr floatType value = type::template get<floatType>();
02403
             };
02404
02405
              template<typename T>
02406
              struct bernoulli<T, 0> {
02407
                 using type = typename FractionField<T>::one;
02408
02409
                  template<typename floatType>
                  static constexpr floatType value = type::template get<floatType>();
02410
02411
              };
02412
         } // namespace internal
02413
02417
          template<typename T, size_t n>
02418
          using bernoulli_t = typename internal::bernoulli<T, n>::type;
02419
          template<typename FloatType, typename T, size_t n >
02424
02425
          inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02426
02427
          // bell numbers
02428
          namespace internal {
              template<typename T, size_t n, typename E = void>
02429
02430
              struct bell_helper;
02431
02432
              template <typename T, size_t n>
02433
              struct bell_helper<T, n, std::enable_if_t<(n > 1)» {
02434
                  template<typename accum, size_t i, size_t stop>
02435
                  struct sum_helper {
02436
                  private:
02437
                      using left = typename T::template mul_t<
                                  combination_t<T, i, n-1>,
02438
02439
                                  typename bell_helper<T, i>::type>;
02440
                      using new_accum = typename T::template add_t<accum, left>;
                   public:
02441
02442
                      using type = typename sum_helper<new_accum, i+1, stop>::type;
02443
02444
02445
                  template<typename accum, size_t stop>
02446
                  struct sum_helper<accum, stop, stop> {
02447
                      using type = accum;
02448
02449
02450
                  using type = typename sum_helper<typename T::zero, 0, n>::type;
02451
              };
02452
02453
              template<typename T>
02454
              struct bell_helper<T, 0> {
02455
                  using type = typename T::one;
02456
02457
              template<typename T>
02458
02459
              struct bell_helper<T, 1> {
02460
                 using type = typename T::one;
02461
              };
02462
          } // namespace internal
02463
02467
          template<typename T, size_t n>
02468
          using bell_t = typename internal::bell_helper<T, n>::type;
02469
          template<typename T, size_t n>
02473
02474
         static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
02475
02476
          namespace internal {
02477
             template<typename T, int k, typename E = void>
02478
              struct alternate {};
02479
02480
              template<typename T, int k>
              struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
02481
                  using type = typename T::one;
02482
02483
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02484
             };
02485
```

```
template<typename T, int k>
02487
               struct alternate<T, k, std::enable_if_t<k % 2 != 0» {</pre>
02488
                   using type = typename T::template sub_t<typename T::zero, typename T::one>;
02489
                   static constexpr typename T::inner_type value = type::template get<typename</pre>
      T::inner_type>();
02490
           } // namespace internal
02491
02492
02495
          template<typename T, int k>
02496
          using alternate_t = typename internal::alternate<T, k>::type;
02497
02498
          namespace internal {
02499
               template<typename T, int n, int k, typename E = void>
02500
               struct stirling_helper {};
02501
02502
               template<typename T>
               struct stirling_helper<T, 0, 0> {
02503
02504
                  using type = typename T::one;
02505
02506
02507
               template<typename T, int n>
02508
               struct stirling_helper<T, n, 0, std::enable_if_t<(n > 0)» {
02509
                  using type = typename T::zero;
02510
02511
02512
               template<typename T, int n>
02513
               struct stirling_helper<T, 0, n, std::enable_if_t<(n > 0)» {
02514
                   using type = typename T::zero;
02515
02516
               template<typename T, int n, int k>
02517
02518
               struct stirling_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)» {
02519
                   using type = typename T::template sub_t<
02520
                                     typename stirling_helper<T, n-1, k-1>::type,
02521
                                     typename T::template mul_t<</pre>
                                         typename T::template inject_constant_t<n-1>,
02522
02523
                                         typename stirling_helper<T, n-1, k>::type
02525
02526
           } // namespace internal
02527
          template<typename T, int n, int k>
02532
          using stirling_signed_t = typename internal::stirling_helper<T, n, k>::type;
02533
02534
02539
           template<typename T, int n, int k>
02540
          using stirling_unsigned_t = abs_t<typename internal::stirling_helper<T, n, k>::type>;
02541
          template<typename T, int n, int k>
static constexpr typename T::inner_type stirling_signed_v = stirling_signed_t<T, n, k>::v;
02546
02547
02548
02549
02554
          template<typename T, int n, int k>
02555
          static constexpr typename T::inner_type stirling_unsigned_v = stirling_unsigned_t<T, n, k>::v;
02556
02559
          template<typename T, size_t k>
02560
          inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
02561
02562
          namespace internal {
02563
               template<typename T>
02564
               struct pow_scalar {
02565
                   template<size t p>
                   static constexpr DEVICE INLINED T func(const T& x) { return p == 0 ? static_cast<T>(1) :
    p % 2 == 0 ? func<p/2>(x) * func<p/2>(x) :
02566
02567
02568
                        x * func < p/2 > (x) * func < p/2 > (x);
02569
                   }
02570
               };
02571
02572
               template<typename T, typename p, size_t n, typename E = void>
02573
               requires IsEuclideanDomain<T>
02574
               struct pow;
02575
02576
               template<typename T, typename p, size_t n>
               struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)» {
02577
02578
                  using type = typename T::template mul_t<
02579
                        typename pow<T, p, n/2>::type,
02580
                        typename pow<T, p, n/2>::type
02581
02582
02583
               template<typename T, typename p, size_t n>
02584
               template typename 1, typename p, size_t m>
struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)» {
    using type = typename T::template mul_t<
02585
02586
02587
02588
                        typename T::template mul_t<
02589
                            typename pow<T, p, n/2>::type,
                            typename pow<T, p, n/2>::type
02590
02591
```

```
>;
02593
02594
02595
               template<typename T, typename p, size_t n>  
02596
               \label{eq:struct_pow} $$ struct pow<T, p, n, std::enable_if_t<n == 0 % { using type = typename T::one; }; $$
02597
          } // namespace internal
02598
02603
          template<typename T, typename p, size_t n>
02604
          using pow_t = typename internal::pow<T, p, n>::type;
02605
02610
          template<typename T, typename p, size_t n>
02611
          static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
02612
02613
          template<typename T, size_t p>
02614
          static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
      internal::pow_scalar<T>::template func(x); }
02615
02616
          namespace internal {
02617
              template<typename, template<typename, size_t> typename, class>
02618
               struct make_taylor_impl;
02619
02620
               template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
02621
               struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...» {</pre>
02622
                  using type = typename polynomial<FractionField<T>::template val<typename coeff_at<T,
      Is>::type...>;
02623
              };
02624
02625
02630
          template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
02631
          using taylor = typename internal::make_taylor_impl<</pre>
02632
02633
               coeff at,
02634
              internal::make_index_sequence_reverse<deg + 1>>::type;
02635
02636
          namespace internal {
              template<typename T, size_t i>
02637
02638
               struct exp coeff {
02639
                   using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02640
02641
02642
               template<typename T, size_t i, typename E = void>
02643
               struct sin_coeff_helper {};
02644
02645
               template<typename T, size_t i>
              struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
02646
02647
02648
02649
               template<typename T, size_t i>
02650
02651
               struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
                  using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02652
02653
02654
02655
               template<typename T, size_t i>
02656
               struct sin coeff {
                  using type = typename sin_coeff_helper<T, i>::type;
02657
02658
02659
02660
               template<typename T, size_t i, typename E = void>
02661
               struct sh_coeff_helper {};
02662
               template<typename T, size_t i>
02663
02664
               struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
                  using type = typename FractionField<T>::zero;
02665
02666
               } ;
02667
02668
               template<typename T, size_t i>
               struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1  {
02669
02670
                   using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
               };
02672
02673
               template<typename T, size_t i>
02674
               struct sh_coeff {
02675
                  using type = typename sh_coeff_helper<T, i>::type;
02676
02677
02678
               template<typename T, size_t i, typename E = void>
02679
               struct cos_coeff_helper {};
02680
02681
               template<typename T. size t i>
               struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
    using type = typename FractionField<T>::zero;
02682
02683
02684
02685
02686
               template<typename T, size_t i>
               struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02687
02688
```

```
02689
              };
02690
02691
              template<typename T, size_t i>
02692
              struct cos_coeff {
02693
                  using type = typename cos_coeff_helper<T, i>::type;
02694
02695
02696
              template<typename T, size_t i, typename E = void>
02697
              struct cosh_coeff_helper {};
02698
02699
              template<typename T, size_t i>
02700
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
                  using type = typename FractionField<T>::zero;
02701
02702
02703
              template<typename T, size_t i>
02704
02705
              struct cosh\_coeff\_helper<T, i, std::enable\_if\_t<(i \& 1) == 0» {
02706
                 using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02708
02709
              template<typename T, size_t i>
02710
              struct cosh_coeff {
02711
                  using type = typename cosh_coeff_helper<T, i>::type;
02712
02713
02714
              template<typename T, size_t i>
02715
              struct geom_coeff { using type = typename FractionField<T>::one; };
02716
02717
02718
              template<typename T, size_t i, typename E = void>
02719
              struct atan coeff helper:
02720
02721
              template<typename T, size_t i>
02722
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1  {
02723
                  using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;;
02724
02725
02726
              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;
02727
02728
02729
02730
              template<typename T, size_t i>
struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
02731
02732
02733
02734
              template<typename T, size_t i, typename E = void>
02735
              struct asin_coeff_helper;
02736
02737
              template<tvpename T, size t i>
02738
              struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02739
                  using type = makefraction_t<T,
02740
                       factorial_t<T, i - 1>,
02741
                       typename T::template mul_t<
02742
                           typename T::template val<i>,
02743
                           T::template mul_t<
02744
                               pow t<T, typename T::template inject constant t<4>, i / 2>,
02745
                               pow<T, factorial_t<T, i / 2>, 2
02746
02747
02748
                       »;
02749
              }:
02750
02751
              template<typename T, size_t i>
02752
              struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
02753
                  using type = typename FractionField<T>::zero;
02754
02755
02756
              template<typename T, size_t i>
02757
              struct asin_coeff {
02758
                  using type = typename asin_coeff_helper<T, i>::type;
02759
02760
02761
              template<typename T, size_t i>
02762
              struct lnp1_coeff {
02763
                  using type = makefraction_t<T,
02764
                       alternate_t<T, i + 1>,
02765
                       typename T::template val<i>;;
02766
02767
02768
              template<tvpename T>
02769
              struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
02771
               template<typename T, size_t i, typename E = void>
02772
              struct asinh_coeff_helper;
02773
02774
              template<typename T, size_t i>
struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
02775
```

```
using type = makefraction_t<T,
02777
                       typename T::template mul_t<
02778
                           alternate_t<T, i / 2>,
                           factorial_t<T, i - 1>
02779
02780
02781
                       typename T::template mul t<
02782
                           typename T::template mul_t<
02783
                                typename T::template val<i>,
02784
                               pow_t<T, factorial_t<T, i / 2>, 2>
02785
02786
                           pow_t<T, typename T::template inject_constant_t<4>, i / 2>
02787
02788
                  >;
02789
              };
02790
               template<typename T, size_t i>
struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
02791
02792
02793
                  using type = typename FractionField<T>::zero;
02795
02796
               template<typename T, size_t i>
02797
               struct asinh_coeff {
02798
                  using type = typename asinh_coeff_helper<T, i>::type;
02799
02800
02801
               template<typename T, size_t i, typename E = void>
02802
               struct atanh_coeff_helper;
02803
02804
               template<typename T, size_t i>
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
    // 1/i</pre>
02805
02806
02807
                   using type = typename FractionField<T>:: template val<</pre>
02808
                       typename T::one,
02809
                       typename T::template inject_constant_t<i>;
02810
02811
02812
               template<typename T, size_t i>
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
02814
                  using type = typename FractionField<T>::zero;
02815
02816
02817
               template<typename T, size_t i>
               struct atanh_coeff {
02818
02819
                   using type = typename atanh_coeff_helper<T, i>::type;
02820
02821
02822
               template<typename T, size_t i, typename E = void>
02823
               struct tan_coeff_helper;
02824
02825
               template<typename T, size_t i>
02826
               struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
02827
                   using type = typename FractionField<T>::zero;
02828
02829
               template<typename T, size_t i>
02830
               struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
02831
02832
               private:
02833
                   // 4^((i+1)/2)
                   using _4p = typename FractionField<T>::template inject_t<</pre>
02834
                   pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»; // 4^{((i+1)/2)} - 1
02835
02836
                   using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
02837
     FractionField<T>::one>;
02838
                  // (-1)^((i-1)/2)
02839
                   using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
02840
                   using dividend = typename FractionField<T>::template mul_t<</pre>
02841
                       altp,
02842
                       FractionField<T>::template mul t<
02843
                       4p.
                       FractionField<T>::template mul_t<
02845
                       _4pm1,
02846
                       bernoulli_t<T, (i + 1)>
02847
02848
02849
02850
               public:
                  using type = typename FractionField<T>::template div_t<dividend,</pre>
02851
02852
                       typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02853
               };
02854
02855
               template<typename T, size_t i>
02856
               struct tan_coeff {
02857
                   using type = typename tan_coeff_helper<T, i>::type;
02858
02859
02860
               template<typename T, size_t i, typename E = void>
02861
               struct tanh_coeff_helper;
```

```
02862
02863
              template<typename T, size_t i>
              struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {</pre>
02864
                 using type = typename FractionField<T>::zero;
02865
02866
02867
02868
              template<typename T, size_t i>
02869
              struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
02870
02871
                  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
02872
02873
     FractionField<T>::one>;
                  using dividend =
02874
02875
                       typename FractionField<T>::template mul_t<</pre>
02876
                           _4p,
                           typename FractionField<T>::template mul_t<</pre>
02877
02878
                               4pm1,
02879
                               bernoulli_t<T, (i + 1) >>::type;
02880
              public:
02881
                  using type = typename FractionField<T>::template div_t<dividend,</pre>
02882
                       FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02883
              };
02884
02885
              template<typename T, size_t i>
02886
              struct tanh_coeff {
02887
                   using type = typename tanh_coeff_helper<T, i>::type;
02888
02889
          } // namespace internal
02890
02894
          template<typename Integers, size_t deg>
02895
          using exp = taylor<Integers, internal::exp_coeff, deg>;
02896
02900
          template<typename Integers, size_t deg>
02901
          using expm1 = typename polynomial<FractionField<Integers>>::template sub_t
02902
              exp<Integers, deg>.
              typename polynomial<FractionField<Integers>>::one>;
02903
02904
02908
          template<typename Integers, size_t deg>
02909
          using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
02910
02914
          template<typename Integers, size_t deg>
02915
          using atan = taylor<Integers, internal::atan coeff, deg>;
02916
02920
          template<typename Integers, size_t deg>
02921
          using sin = taylor<Integers, internal::sin_coeff, deg>;
02922
02926
          template<typename Integers, size_t deg>
02927
          using sinh = taylor<Integers, internal::sh_coeff, deg>;
02928
02933
          template<typename Integers, size_t deg>
02934
          using cosh = taylor<Integers, internal::cosh_coeff, deg>;
02935
02940
          template<typename Integers, size_t deg>
02941
          using cos = taylor<Integers, internal::cos_coeff, deg>;
02942
02947
          template<typename Integers, size_t deg>
02948
          using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
02949
02954
          template<typename Integers, size_t deg>
          using asin = taylor<Integers, internal::asin_coeff, deg>;
02955
02956
02961
          template<typename Integers, size_t deg>
02962
          using asinh = taylor<Integers, internal::asinh_coeff, deg>;
02963
02968
          template<typename Integers, size_t deg>
02969
          using atanh = taylor<Integers, internal::atanh_coeff, deg>;
02970
02975
          template<typename Integers, size_t deg>
02976
          using tan = taylor<Integers, internal::tan_coeff, deg>;
02977
02982
          template<typename Integers, size_t deg>
02983
          using tanh = taylor<Integers, internal::tanh_coeff, deg>;
02984 } // namespace aerobus
02985
02986 // continued fractions
02987 namespace aerobus {
02990
          template<int64_t... values>
02991
          struct ContinuedFraction {};
02992
02995
          template<int64 t a0>
          struct ContinuedFraction<a0> {
02998
              using type = typename q64::template inject_constant_t<a0>;
03000
              static constexpr double val = static_cast<double>(a0);
03001
          };
03002
03006
          template<int64 t a0, int64 t... rest>
```

```
struct ContinuedFraction<a0, rest...> {
03009
                       using type = q64::template add_t<
03010
                                      typename q64::template inject_constant_t<a0>,
03011
                                     typename q64::template div_t <
03012
                                            typename q64::one,
typename ContinuedFraction<rest...>::type
03013
03014
03015
                       static constexpr double val = type::template get<double>();
03017
03018
                 };
03019
                using PI_fraction =
03024
          ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1>;
03027
                 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>;
03029
                using SQRT2_fraction =
          03031
                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, 
03032 } // namespace aerobus
03033
03034 // known polynomials
03035 namespace aerobus {
03036
                 // CChebyshev
03037
                 namespace internal {
                        template<int kind, size_t deg, typename I>
03038
03039
                        struct chebyshev_helper {
03040
                              using type = typename polynomial<I>::template sub_t<</pre>
03041
                                     typename polynomial<I>::template mul_t<</pre>
                                            typename polynomial<I>::template mul_t<</pre>
03042
03043
                                                   typename polynomial<I>::template inject_constant_t<2>,
03044
                                                    typename polynomial<I>::X>,
03045
                                            typename chebyshev_helper<kind, deg - 1, I>::type
03046
                                     typename chebyshev_helper<kind, deg - 2, I>::type
03047
03048
                              >;
03049
                        } ;
03050
03051
                        template<typename I>
03052
                        struct chebyshev_helper<1, 0, I> {
                              using type = typename polynomial<I>::one;
03053
03054
03055
03056
                        template<typename I>
03057
                        struct chebyshev_helper<1, 1, I> {
03058
                              using type = typename polynomial<I>::X;
03059
                        };
03060
03061
                        template<tvpename I>
03062
                        struct chebyshev_helper<2, 0, I> {
03063
                              using type = typename polynomial<I>::one;
03064
03065
03066
                        template<typename I>
03067
                        struct chebyshev_helper<2, 1, I> {
                              using type = typename polynomial<I>::template mul_t<
03068
03069
                                     typename polynomial<I>::template inject_constant_t<2>,
03070
                                     typename polynomial<I>::X>;
03071
                 } // namespace internal
03072
03073
03074
                 // Laguerre
03075
                 namespace internal {
03076
                        template<size_t deg, typename I>
03077
                        struct laguerre_helper {
                             using Q = FractionField<I>;
using PQ = polynomial<Q>;
03078
03079
03080
03081
                          private:
03082
                              // Lk = (1 / k) * ((2 * k - 1 - x) * 1km1 - (k - 2)Lkm2)
03083
                              using lnm2 = typename laguerre_helper<deg - 2, I>::type;
                              using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03084
03085
                               // -x + 2k-1
03086
                               using p = typename PO::template val<
03087
                                     typename Q::template inject_constant_t<-1>,
03088
                                     typename Q::template inject_constant_t<2 * deg - 1»;
03089
                               // 1/n
03090
                               using factor = typename PQ::template inject_ring_t<
03091
                                     \verb|typename Q::template val<typename I::one, typename I::template|\\
          inject_constant_t<deg>>;
03092
03093
                          public:
03094
                              using type = typename PQ::template mul_t <
03095
                                     factor,
                                      typename PQ::template sub_t<
03096
03097
                                            typename PO::template mul t<
```

```
03098
03099
03100
0.3101
                          typename PQ::template mul_t<
03102
                              typename PQ::template inject_constant_t<deg-1>,
                              lnm2
03103
03104
03105
03106
                 >;
03107
              } ;
03108
03109
              template<tvpename I>
03110
              struct laguerre_helper<0, I> {
03111
                 using type = typename polynomial<FractionField<I>::one;
03112
03113
03114
              template<typename I>
              struct laguerre_helper<1, I> {
03115
03116
              private:
03117
                 using PQ = polynomial<FractionField<I>;
03118
               public:
03119
                 using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
03120
03121
         } // namespace internal
03122
03123
          // Bernstein
03124
         namespace internal {
03125
              template<size_t i, size_t m, typename I, typename E = void>
03126
              struct bernstein_helper {};
03127
03128
              template<tvpename I>
03129
             struct bernstein_helper<0, 0, I> {
03130
                 using type = typename polynomial<I>::one;
03131
              };
03132
              template<size_t i, size_t m, typename I>
03133
             struct bernstein_helperi, m, I, std::enable_if_t<
(m > 0) && (i == 0) » {
03134
03135
03136
              private:
03137
                 using P = polynomial<I>;
03138
               public:
03139
                 using type = typename P::template mul_t<
                          typename P::template sub_t<typename P::one, typename P::X>,
03140
03141
                          typename bernstein_helper<i, m-1, I>::type>;
03142
              } ;
03143
03144
              template<size_t i, size_t m, typename I>
              03145
03146
03147
               private:
03148
                 using P = polynomial<I>;
03149
               public:
03150
                 using type = typename P::template mul_t<
03151
                          typename P::X,
                          typename bernstein_helper<i-1, m-1, I>::type>;
03152
03153
              };
03154
03155
              template<size_t i, size_t m, typename I>
03156
              struct bernstein_helper<i, m, I, std::enable_if_t<
                          (m > 0) && (i > 0) && (i < m)» {
03157
03158
               private:
                 using P = polynomial<I>;
03159
03160
               public:
03161
                 using type = typename P::template add_t<
03162
                          typename P::template mul_t<
03163
                              typename P::template sub_t<typename P::one, typename P::X>,
03164
                              typename bernstein_helper<i, m-1, I>::type>,
                          typename P::template mul_t<
03165
03166
                              typename P::X,
03167
                              typename bernstein_helper<i-1, m-1, I>::type»;
03168
03169
          } // namespace internal
03170
03171
         namespace known_polynomials {
03173
              enum hermite kind {
                 probabilist,
03175
                 physicist
03177
03178
             } ;
03179
         }
0.3180
          // hermite
03181
03182
         namespace internal {
03183
             template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03184
              struct hermite_helper {};
03185
03186
              template<size_t deg, typename I>
              struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
03187
```

```
03188
               private:
                  using hnm1 = typename hermite_helper<deg - 1,
03189
      known_polynomials::hermite_kind::probabilist, I>::type;
03190
                 using hnm2 = typename hermite_helper<deg - 2,
     known_polynomials::hermite_kind::probabilist, I>::type;
03191
               public:
03192
03193
                  using type = typename polynomial<I>::template sub_t<
03194
                      typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnml>,
03195
                      typename polynomial<I>::template mul_t<</pre>
03196
                          typename polynomial<I>::template inject_constant_t<deg - 1>,
03197
                          hnm2
03198
03199
                  >;
03200
              };
03201
03202
              template<size_t deg, typename I>
03203
              struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03204
03205
                  using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
     I>::tvpe;
03206
                  using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
     I>::type;
03207
03208
               public:
03209
                  using type = typename polynomial<I>::template sub_t<</pre>
03210
                       // 2X Hn-1
03211
                      typename polynomial<I>::template mul_t<
03212
                          typename pi64::val<typename I::template inject_constant_t<2>,
03213
                          typename I::zero>, hnml>,
03214
03215
                      typename polynomial<I>::template mul_t<</pre>
03216
                          typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03217
                          hnm2
03218
03219
                  >;
03220
              };
03221
03222
              template<typename I>
03223
              struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03224
                  using type = typename polynomial<I>::one;
03225
              };
03226
03227
              template<typename I>
              struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03228
                  using type = typename polynomial<I>::X;
03229
03230
03231
03232
              template<tvpename I>
03233
              struct hermite helper<0, known polynomials::hermite kind::physicist, I> {
03234
                  using type = typename pi64::one;
03235
03236
03237
              template<typename I>
03238
              struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03239
                  // 2X
03240
                  using type = typename polynomial<I>::template val<
03241
                      typename I::template inject_constant_t<2>,
03242
                      typename I::zero>;
03243
         } // namespace internal
03244
03245
03246
          // legendre
03247
          namespace internal {
03248
              template<size_t n, typename I>
03249
              struct legendre_helper {
               private:
03250
03251
                  using O = FractionField<I>;
03252
                  using PQ = polynomial<Q>;
03253
                  // 1/n constant
03254
                  // (2n-1)/n X
03255
                  using fact_left = typename PQ::template monomial_t<</pre>
                      makefraction_t<I,
03256
03257
                          typename I::template inject_constant_t<2*n-1>,
03258
                          typename I::template inject_constant_t<n>
03259
03260
                  1>;
03261
                  // (n-1) / n
03262
                  using fact_right = typename PQ::template val<
                      makefraction_t<I,
03263
03264
                         typename I::template inject constant t<n-1>,
03265
                          typename I::template inject_constant_t<n>>;
03266
03267
               public:
03268
                  using type = PQ::template sub_t<
                          typename PQ::template mul_t<
03269
03270
                              fact left.
```

```
typename legendre_helper<n-1, I>::type
03272
03273
                            typename PQ::template mul_t<
03274
                                fact_right,
03275
                                typename legendre_helper<n-2, I>::type
03276
03277
                        >;
03278
03279
03280
               template<typename I>
               struct legendre_helper<0, I> {
03281
                  using type = typename polynomial<FractionField<I»::one;</pre>
03282
03283
03284
03285
               template<typename I>
03286
               struct legendre_helper<1, I> {
03287
                   using type = typename polynomial<FractionField<I>::X;
03288
03289
          } // namespace internal
03290
03291
           // bernoulli polynomials
03292
          namespace internal {
03293
               template<size_t n>
               struct bernoulli coeff {
03294
03295
                   template<typename T, size_t i>
                   struct inner {
03296
03297
                    private:
03298
                       using F = FractionField<T>;
03299
                    public:
03300
                       using type = typename F::template mul t<
03301
                            typename F::template inject_ring_t<combination_t<T, i, n»,
03302
                            bernoulli_t<T, n-i>
03303
03304
                   };
03305
          } // namespace internal
03306
03307
03309
          namespace known_polynomials {
03317
               template <size_t deg, typename I = aerobus::i64>
03318
               using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
03319
03329
               template <size_t deg, typename I = aerobus::i64>
03330
               using chebyshev U = typename internal::chebyshev helper<2, deg, I>::type;
03331
03341
               template <size_t deg, typename I = aerobus::i64>
03342
               using laguerre = typename internal::laguerre_helper<deg, I>::type;
03343
03350
               template <size_t deg, typename I = aerobus::i64>
              using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,</pre>
03351
      I>::tvpe;
03352
03359
               template <size_t deg, typename I = aerobus::i64>
03360
               using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
03361
               template<size_t i, size_t m, typename I = aerobus::i64>
03372
03373
               using bernstein = typename internal::bernstein helper<i, m, I>::type;
03374
03384
               template<size_t deg, typename I = aerobus::i64>
03385
               using legendre = typename internal::legendre_helper<deg, I>::type;
03386
03396
               template<size_t deg, typename I = aerobus::i64>
03397
              using bernoulli = taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
03398
             // namespace known_polynomials
03399 } // namespace aerobus
03400
03401
03402 #ifdef AEROBUS CONWAY IMPORTS
03403
03404 // conway polynomials
03405 namespace aerobus {
03409
         template<int p, int n>
03410
          struct ConwayPolynomial {};
03411
03412 #ifndef DO NOT DOCUMENT
          #define ZPZV ZPZ::template val
03413
           #define POLYV aerobus::polynomial<ZPZ>::template val
03414
           template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
03415
      POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
          template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
03416
      POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
          template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
03417
      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>; ZPZV<1); / NOLINT
03419
          template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
03420
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<1>, ZPZV<1»; };
                                 template<> struct ConwayPolynomial<2, 7> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; // NOL
                                                       template<> struct ConwayPolynomial<2, 8> { using ZPZ = aerobus::zpz<2>; using type =
 03422
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1»; };
                                                          template<> struct ConwayPolynomial<2, 9> { using ZPZ = aerobus::zpz<2>; using type
03423
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; //
                                                        template<> struct ConwayPolynomial<2, 10> { using ZPZ = aerobus::zpz<2>; using type =
03424
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>,
                                   ZPZV<1»; }; // NOLINT</pre>
                                                        template<> struct ConwayPolynomial<2, 11> { using ZPZ = aerobus::zpz<2>; using type
03425
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                                      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
                                                        template<> struct ConwayPolynomial<2, 13> { using ZPZ = aerobus::zpz<2>; using type
03427
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , Z
                                  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<0>,
03429
                                   ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                                       template<> struct ConwayPolynomial<2, 16> { using ZPZ = aerobus::zpz<2>; using type =
03430
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   03431
                                  template<> struct ConwayPolynomial<2, 17> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
                                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                                          template<> struct ConwayPolynomial<2, 18> { using ZPZ = aerobus::zpz<2>; using type =
03432
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>,
                                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>; }; // NOLINT
03433
                                  template<> struct ConwayPolynomial<2, 19> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
                                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
                                  template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                    ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1</pre>; };
                                    // NOLINT
                                                          template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
                                  POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
 03436
                                                          template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
                                  POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2»; }; // NOLINT
 03437
                                                        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 =
 03438
                                 POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; }; // NOLINT
                                                          template<> struct ConwayPolynomial<3, 5> { using ZPZ = aerobus::zpz<3>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT
 03440
                                                        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»; }; // NOLINT
                                                          template<> struct ConwayPolynomial<3, 7> { using ZPZ = aerobus::zpz<3>; using type
 03441
                                   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 =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2»; }; // NOLINT
 03443
                                                           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 =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
                                   ZPZV<2»; }; // NOLINT</pre>
03445
                                                      template<> struct ConwayPolynomial<3, 11> { using ZPZ = aerobus::zpz<3>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1»; }; // NOLINT
                                                        template<> struct ConwayPolynomial<3, 12> { using ZPZ = aerobus::zpz<3>; using type =
                                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                                      template<> struct ConwayPolynomial<3, 13> { using ZPZ = aerobus::zpz<3>; using type =
03447
                                   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<2>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<3, ZPZV<
03448
                                   ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT</pre>
                                                          template<> struct ConwayPolynomial<3, 15> { using ZPZ = aerobus::zpz<3>; using type =
                                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0 , ZPZ
03450
                                   ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
 03451
                                                          template<> struct ConwayPolynomial<3, 17> { using ZPZ = aerobus::zpz<3>; using type =
                                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
 03452
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                           ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3, ZPZV<3
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; //</pre>
                            NOLINT
                           template<> struct ConwayPolynomial<3, 20> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>,
                             ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1
, ZPZV<1
                            // NOLINT
03455
                                              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 =
03456
                           POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2»; }; // NOLINT
03457
                                           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 =
03458
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; };
                                                                                                                                                                                                                                                                                        // NOLINT
                                              template<> struct ConwayPolynomial<5, 5> { using ZPZ = aerobus::zpz<5>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<3»; }; // NOLINT
                                               template<> struct ConwayPolynomial<5, 6> { using ZPZ = aerobus::zpz<5>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT
                                            template<> struct ConwayPolynomial<5, 7> { using ZPZ = aerobus::zpz<5>; using type =
03461
                            \texttt{POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3»; }; \ // \ \texttt{NOLINT}  
03462
                                              template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                            template<> struct ConwayPolynomial<5, 9> { using ZPZ = aerobus::zpz<5>; using type
03463
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<3»; }; //
                            NOLINT
                                            template<> struct ConwayPolynomial<5, 10> { using ZPZ = aerobus::zpz<5>; using type =
03464
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<1>,
                            ZPZV<2»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<5, 11> { using ZPZ = aerobus::zpz<5>; using type =
03465
                            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
03466
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<2»; }; // NOLINT
                                                template<> struct ConwayPolynomial<5, 13> { using ZPZ = aerobus::zpz<5>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<3»; }; // NOLINT
                           template<> struct ConwayPolynomial<5, 14> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5, 
03468
                            ZPZV<2>, ZPZV<3>, ZPZV<0>, ZPZV<1>, ZPZV<2»; }; // NOLINT
                                              template<> struct ConwayPolynomial<5, 15> { using ZPZ = aerobus::zpz<5>; using type
                           POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                            ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<3»; }; // NOLINT
                                              template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>,
                            ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<5, 17> { using ZPZ = aerobus::zpz<5>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<3»; }; // NOLINT</pre>
03472
                           template<> struct ConwayPolynomial<5, 18> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>,
                            ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<5, 19> { using ZPZ = aerobus::zpz<5>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<3»; }; //</pre>
                            NOLINT
03474
                           template<> struct ConwayPolynomial<5, 20> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
                             ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<0>, ZPZV<4>, ZPZV<1>, ZPZV<1>, ZPZV<2»; };</pre>
                                            template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
03475
                           POLYV<ZPZV<1>, ZPZV<4»; }; // NOLINT
03476
                                              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 =
                         POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4»; ); // NOLINT template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type =
03478
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; }; // NOLINT
                                              template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
03479
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT
                                              template<> struct ConwayPolynomial<7, 6> { using ZPZ = aerobus::zpz<7>; using type =
03480
                           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
03481
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<4»; }; // NOLINT
03482
                                            template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<6>, ZPZV<2>, ZPZV<3»; }; // NOLINT
                           template<> struct ConwayPolynomial<7, 9> { using ZPZ = aerobus::2pz<7>; using type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZP
03483
03484
                                            template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<3»; }; // NOLINT
03485
                                           template<> struct ConwayPolynomial<7, 11> { using ZPZ = aerobus::zpz<7>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                          ZPZV<1>, ZPZV<4»; }; // NOLINT</pre>
                                                                  template<> struct ConwayPolynomial<7, 12> { using ZPZ = aerobus::zpz<7>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<5>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
                                                                    template<> struct ConwayPolynomial<7, 13> { using ZPZ = aerobus::zpz<7>; using type =
03487
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<0>, ZPZV<6>, ZPZV<0>, ZPZV<4»; }; // NOLINT</pre>
                                                                  template<> struct ConwayPolynomial<7, 14> { using ZPZ = aerobus::zpz<7>; using type =
03488
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<6>,
                                        ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<6>, ; // NOLINT
    template<> struct ConwayPolynomial<7, 15> { using ZPZ = aerobus::zpz<7>; using type
03489
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<6>, ZPZV<6>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<4»; }; // NOLINT
                                                                template<> struct ConwayPolynomial<7, 16> { using ZPZ = aerobus::zpz<7>; using type
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4 , ZPZV<4
                                        template<> struct ConwayPolynomial</pr>
7, 17> { using ZPZ = aerobus::zpz<f>; using type = POLYV<ZPZV<1>, ZPZV<0>, 
03491
                                         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<6>, ZPZV<5, ZPZV<6>, ZPZV<5, ZPZV<6>, ZPZV<6 , ZPZ
                                        ZPZV<6>, ZPZV<5>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<2>, ZPZV<2>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<7, 19> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
03493
                                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<4»; }; //</pre>
                                        template<> struct ConwayPolynomial<7, 20> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<6>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3</pre>
03494
                                         // NOLINT
03495
                                                                     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 =
03496
                                        POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2»; }; // NOLINT
                                                                     template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
03497
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<9»; }; // 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
 03499
                                                                  template<> struct ConwayPolynomial<11, 5> { using ZPZ = aerobus::zpz<11>; using type =
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<0>, ZPZV<9»; }; // NOLINT
                                        template<> struct ConwayPolynomial<11, 6> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<6>, ZPZV<7>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<11, 7> { using ZPZ = aerobus::zpz<11>; using type =
 03500
 03501
                                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<9»; }; // NOLINT
 03502
                                                                    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
03503
                                                                template<> struct ConwayPolynomial<11, 9> { using ZPZ = aerobus::zpz<11>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<8>, ZPZV<9»; }; //
                                         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>,
                                         ZPZV<2»; }; // NOLINT</pre>
03505
                                                                    template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                     template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type
                                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<2»; }; // NOLINT
03507
                                                                     template<> struct ConwayPolynomial<11, 13> { using ZPZ = aerobus::zpz<11>; using type
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03508
                                                                    template<> struct ConwayPolynomial<11, 14> { using ZPZ = aerobus::zpz<11>; using type
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<5>, ZPZV<6>,
                                          ZPZV<4>, ZPZV<8>, ZPZV<6>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
03509
                                                                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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZV<0 , ZPZ
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZ
                                          ZPZV<1>, ZPZV<3>, ZPZV<5>, ZPZV<3>, ZPZV<10>, ZPZV<9>, ZPZV<2»; }; // NOLINT</pre>
                                                                template<> struct ConwayPolynomial<11, 17> { using ZPZ = aerobus::zpz<11>; using type =
03511
                                         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<8>, ZPZV<8-, ZPZV<8-,
03512
                                         ZPZV<3>, ZPZV<9>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<9>, ZPZV<8>, ZPZV<2>, ZPZV<2»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<11, 19> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03513
                                          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<0>, ZPZV<1>, ZPZV<10>, ZPZV<1>, ZPZV<10>, ZPZV<10
                                          ZPZV<9>, ZPZV<1>, ZPZV<5>, ZPZV<7>, ZPZV<2>, ZPZV<4>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<5</pre>; };
                                         // NOLINT
                                                                   template<> struct ConwayPolynomial<13, 1> { using ZPZ = aerobus::zpz<13>; using type =
                                         POLYV<ZPZV<1>, ZPZV<11»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<13, 2> { using ZPZ = aerobus::zpz<13>; using type =
                         POLYV<ZPZV<1>, ZPZV<12>, ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<13, 3> { using ZPZ = aerobus::zpz<13>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11»; }; // NOLINT

template<> struct ConwayPolynomial<13, 4> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2»; }; // NOLINT
 03518
                                          template<> struct ConwayPolynomial<13, 5> { using ZPZ = aerobus::zpz<13>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<11»; }; // NOLINT
 03520
                                        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 =
03521
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<11»; }; // NOLINT
03522
                                          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»; };
                                       template<> struct ConwayPolynomial<13, 9> { using ZPZ = aerobus::zpz<13>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<8>, ZPZV<12>, ZPZV<12>, ZPZV<11»; };
                          // NOLINT
                         template<> struct ConwayPolynomial<13, 10> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<5>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>,
03524
                          ZPZV<2»: }: // NOLINT
                         template<> struct ConwayPolynomial<13, 11> { using ZPZ = aerobus::zpz<13>; using type = 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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<8>, ZPZV<11>, ZPZV<3>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<2»; }; // NOLINT
03526
                                        template<> struct ConwayPolynomial<13, 13> { using ZPZ = aerobus::zpz<13>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<11»; }; // NOLINT
  template<> struct ConwayPolynomial<13, 14> { using ZPZ = aerobus::zpz<13>; using type =
03528
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<11>, ZPZV<7>, ZPZV<10>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type =
03529
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>,
                          ZPZV<2>, ZPZV<11>, ZPZV<10>, ZPZV<11>, ZPZV<8>, ZPZV<11»; }; // NOLINT
template<> struct ConwayPolynomial<13, 16> { using ZPZ = aerobus::zpz<13>; using type :
03530
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<12>, ZPZV<8>, ZPZV<2>, ZPZV<2>, ZPZV<12>, ZPZV<6>, ZPZV<6>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<13, 17> { using ZPZ = aerobus::zpz<13>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 10>, \ \texttt{ZPZV} < 6>, \ \texttt{ZPZV} < 11»; \ \ \}; \ \ \ // \ \ \texttt{NOLINT} 
                         template<> struct ConwayPolynomial<13, 18> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<11>,
03532
                          ZPZV<11>, ZPZV<9>, ZPZV<5>, ZPZV<3>, ZPZV<5>, ZPZV<6>, ZPZV<0>, ZPZV<9>, ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<13, 19> { using ZPZ = aerobus::zpz<13>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0</pre>
                          NOLINT
03534
                                         template<> struct ConwayPolynomial<13, 20> { using ZPZ = aerobus::zpz<13>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                           ZPZV<9>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<7>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5</pre>; };
03535
                                          template<> struct ConwayPolynomial<17, 1> { using ZPZ = aerobus::zpz<17>; using type =
                         POLYV<ZPZV<1>, ZPZV<14»; }; // NOLINT
                                         template<> struct ConwayPolynomial<17, 2> { using ZPZ = aerobus::zpz<17>; using type =
03536
                         POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»; }; // NOLINT
                                           template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
                         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 =
03539
 03540
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<10>, ZPZV<3>, ZPZV<3»; }; // NOLINT
03541
                                        template<> struct ConwayPolynomial<17, 7> { using ZPZ = aerobus::zpz<17>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1, ZPZV<1
 03542
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<14»; };
03544
                                       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
                         template<> struct ConwayPolynomial<17, 11> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03545
                          ZPZV<5>, ZPZV<14»; };</pre>
                                                                                                                         // NOLINT
                                         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<6>, ZPZV<14>, ZPZV<9>, ZPZV<9
                                           template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type
03547
                           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<15>, ZPZV<14*; }; // NOLINT</pre>
                                        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<1>, ZPZV<11>, ZPZV<11>, ZPZV<1>, ZPZV<8>,
                         ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::zpz<17>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>,
                         ZPZV<4>, ZPZV<16>, ZPZV<65, ZPZV<14>, ZPZV<14>, ZPZV<14»; }; // NOLINT
template<> struct ConwayPolynomial<17, 16> { using ZPZ = aerobus::zpz<17>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<13>,
                        ZPZV<5>, ZPZV<2>, ZPZV<12>, ZPZV<13>, ZPZV<12>, ZPZV<1>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<17, 17> { using ZPZ = aerobus::zpz<17>; using type =
03551
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         template<> struct ConwayPolynomial<17, 18> { using ZPZ = aerobus::zpz<17>; using type =
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
03553
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<14»; }; //</pre>
                         NOLINT
                        template<> struct ConwayPolynomial<17, 20> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1</pre>
                          ZPZV<3»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<19, 1> { using ZPZ = aerobus::zpz<19>; using type =
                        POLYV<ZPZV<1>, ZPZV<17»; };
                                                                                                                                                // NOLINT
                                         template<> struct ConwayPolynomial<19, 2> { using ZPZ = aerobus::zpz<19>; using type =
03556
                        POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<19, 3> { using ZPZ = aerobus::zpz<19>; using type =
03557
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17»; }; // NOLINT
                                          template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2»; }; // NOLINT
03559
                                         template<> struct ConwayPolynomial<19, 5> { using ZPZ = aerobus::zpz<19>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17»; }; // NOLINT
                       template<> struct ConwayPolynomial<19, 6> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<6>, ZPZV<27>, ZPZV<29; }; // NOLINT template<> struct ConwayPolynomial<19, 7> { using ZPZ = aerobus::zpz<19>; using type =
03560
03561
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<17»; }; // NOLINT
                                      template<> struct ConwayPolynomial<19, 8> { using ZPZ = aerobus::zpz<19>; using type =
03562
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<10>, ZPZV<3>, ZPZV<2»; }; template<> struct ConwayPolynomial<19, 9> { using ZPZ = aerobus::zpz<19>; using type =
03563
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<14>, ZPZV<16>, ZPZV<17»; };
                                         template<> struct ConwayPolynomial<19, 10> { using ZPZ = aerobus::zpz<19>; using type
                         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<17»; }; // NOLINT</pre>
03565
                                         template<> struct ConwayPolynomial<19, 12> { using ZPZ = aerobus::zpz<19>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<18>, ZPZV<2>, ZPZV<9>,
                         ZPZV<16>, ZPZV<7>, ZPZV<2>; }; // NOLINT
                                        template<> struct ConwayPolynomial<19, 13> { using ZPZ = aerobus::zpz<19>, using type =
                        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<1>, ZPZV<11>, ZPZV<11>, ZPZV<11>,
                        ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<7>, ZPZV<2»; }; // NOLINT
  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<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
03569
                        ZPZV<11>, ZPZV<13>, ZPZV<15>, ZPZV<14>, ZPZV<0>, ZPZV<17»; }; // NOLINT
   template<> struct ConwayPolynomial<19, 16> { using ZPZ = aerobus::zpz<19>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<13>, ZPZV<0>, ZPZV<15>, ZPZV<9>, ZPZV<6>, ZPZV<14>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<19, 17> { using ZPZ = aerobus::zpz<19>; using type
03571
                        POLYYCZPZVC1>, ZPZVC0>, ZPZVC0
                                         template<> struct ConwayPolynomial<19, 18> { using ZPZ = aerobus::zpz<19>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<2
                          \texttt{ZPZV} < 17>, \ \texttt{ZPZV} < 5>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 16>, \ \texttt{ZPZV} < 5>, \ \texttt{ZPZV} < 7>, \ \texttt{ZPZV} < 3>, \ \texttt{ZPZV} < 14>, \ \texttt{ZPZV} < 2»; \ \}; 
03573
                                      template<> struct ConwayPolynomial<19, 19> { using ZPZ = aerobus::zpz<19>; using type =
                        POLYV-ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                          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<5>, ZPZV<6>, ZPZV<6 - , ZPZ
                         }; // NOLINT
03575
                                          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 =
                        POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; }; // NOLINT
                                         template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; }; // NOLINT template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
03578
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5; }; // NOLINT template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<18»; }; // NOLINT
03580
                                       template<> struct ConwayPolynomial<23, 6> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<9>, ZPZV<1>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<23, 7> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<18»; }; // NOLINT
03581
```

```
template<> struct ConwayPolynomial<23, 8> { using ZPZ = aerobus::zpz<23>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5»; }; // NOLINT
                                   template<> struct ConwayPolynomial<23, 9> { using ZPZ = aerobus::zpz<23>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<18»; };
                      // NOLINT
                                     template<> struct ConwayPolynomial<23, 10> { using ZPZ = aerobus::zpz<23>; using type =
03584
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<15>, ZPZV<15>, ZPZV<15>, ZPZV<6>, ZPZV<1>,
                       ZPZV<5»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<23, 11> { using ZPZ = aerobus::zpz<23>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<7>, ZPZV<18»; }; // NOLINT
                      template<> struct ConwayPolynomial<23, 12> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<21>, ZPZV<21>, ZPZV<15>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<15>; // NOLINT
03586
                                  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
                                    template<> struct ConwayPolynomial<23, 14> { using ZPZ = aerobus::zpz<23>; using type
03588
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<1>,
                      ZPZV<18>, ZPZV<19>, ZPZV<1>, ZPZV<22>, ZPZV<5»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<23, 15> { using ZPZ = aerobus::zpz<23>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
03590
                      ZPZV<19>, ZPZV<16>, ZPZV<13>, ZPZV<14>, ZPZV<17>, ZPZV<5>; }; // NOLINT
template<> struct ConwayPolynomial<23, 17> { using ZPZ = aerobus::zpz<23>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                      template<> struct ConwayPolynomial<23, 18> { using ZPZ = aerobus::zpz<23>; using type =
03592
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<18>, ZPZV<2>, ZPZV<1>,
                      ZPZV<18>, ZPZV<3>, ZPZV<16>, ZPZV<21>, ZPZV<0>, ZPZV<11>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<23, 19> { using ZPZ = aerobus::zpz<23>; using type
03593
                      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<5>, ZPZV<5>, ZPZV<18»; }; //</pre>
                      NOLINT
03594
                                    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 =
                      POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; }; // NOLINT
03596
                                   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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<2»; }; // NOLINT
 03597
                                     template<> struct ConwayPolynomial<29, 5> { using ZPZ = aerobus::zpz<29>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<27»; }; // NOLINT
 03599
                                  template<> struct ConwayPolynomial<29, 6> { using ZPZ = aerobus::zpz<29>; using type =
                    POLYV<2P2V<1>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<17>, ZPZV<13>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type =
 03600
                      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<27»; }; // NOLINT
 03601
                                     template<> struct ConwayPolynomial<29, 8> { using ZPZ = aerobus::zpz<29>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<23>; };
                      NOLINT
                      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<20>, ZPZV<22>, ZPZV<22>, ZPZV<22>, ZPZV<27*); };
03602
                       // NOLINT
                                     template<> struct ConwayPolynomial<29, 10> { using ZPZ = aerobus::zpz<29>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<8>, ZPZV<17>, ZPZV<17>, ZPZV<22>,
                       ZPZV<2»; }; // NOLINT</pre>
03604
                                     template<> struct ConwayPolynomial<29, 11> { using ZPZ = aerobus::zpz<29>; using type
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<8>, ZPZV<27»; }; // NOLINT
03605
                                     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<9>, ZPZV<16>, ZPZV<25>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2»; }; // NOLINT
03606
                                  template<> struct ConwayPolynomial<29, 13> { using ZPZ = aerobus::zpz<29>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    template<> struct ConwayPolynomial<29, 14> { using ZPZ = aerobus::zpz<29>; using type =
03607
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<14>, ZPZV<10>,
                       ZPZV<21>, ZPZV<18>, ZPZV<27>, ZPZV<5>, ZPZV<2»; }; // NOLINT</pre>
03608
                                  template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::zpz<29>; using type =
                      Template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::ZPZV29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<13>, ZPZV<14>, ZPZV<8>, ZPZV<12>, 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 , ZPZ
03609
                      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>, ZPZ
03610
                      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<2>, ZPZV<24>, ZPZV<24>, ZPZV<1>, ZPZV<1 , ZPZV<
03611
                      ZPZV<6>, ZPZV<26>, ZPZV<20>, 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<4>, ZPZV<27»; }; //</pre>
                      NOLTNT
```

```
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
03615
                                                 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
03617
                                                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 =
03618
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<16>, ZPZV<8>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<28»; }; // NOL
03619
                                               template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>, ZPZV<24>, ZPZV<3»; };
                             NOLINT
                             template<> struct ConwayPolynomial<31, 9> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>,
03621
                             template<> struct ConwayPolynomial<31, 10> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<26>, ZPZV<13>, ZPZV<13
                              ZPZV<3»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<31, 11> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03623
                              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<28>, ZPZV<2>, ZPZV<9>, ZPZV<25>, ZPZV<12>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<31, 13> { using ZPZ = aerobus::zpz<31>; using type =
03625
                             POLYY<ZPZV<0>, ZPZV<0>, ZPZV<0
                                                  template<> struct ConwayPolynomial<31, 14> { using ZPZ = aerobus::zpz<31>; using type =
03626
                              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

template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZ
03627
                                                   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
    template<> struct ConwayPolynomial<31, 17> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03629
                             ZPZV<0>, ZPZV<1>); // NOLINT 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<7>, ZPZV<12>, ZPZV<11>, ZPZV<25>, ZPZV<10>, ZPZV<10>, ZPZV<3); }; // NOLINT template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3, 
                              NOLINT
                                                   template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
                             POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                                template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
03633
                            POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
03634
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT
                                                template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<37, 5> { using ZPZ = aerobus::zpz<37>; using type =
03636
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV
03637
                                                    template<> struct ConwayPolynomial<37, 7> { using ZPZ = aerobus::zpz<37>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<35»; };
03639
                                                template<> struct ConwayPolynomial<37, 8> { using ZPZ = aerobus::zpz<37>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<2>, ZPZV<20>, ZPZV<27>, ZPZV<27>, ZPZV<2*; };
                                                template<> struct ConwayPolynomial<37, 9> { using ZPZ = aerobus::zpz<37>; using type
03640
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<20>, ZPZV<32>, ZPZV<35»; };
                              // NOLINT
                                                  template<> struct ConwayPolynomial<37, 10> { using ZPZ = aerobus::zpz<37>; using type
03641
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<11>, ZPZV<41,
                               ZPZV<2»; }; // NOLINT</pre>
                                                  template<> struct ConwayPolynomial<37, 11> { using ZPZ = aerobus::zpz<37>; using type =
03642
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<10>, ZPZV<23>, ZPZV<23>, ZPZV<18>, ZPZV<33>, ZPZV<28>; // NOLINT
03644
                                                template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                template<> struct ConwayPolynomial<37, 14> { using ZPZ = aerobus::zpz<37>; using type
                             POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<1>, 2PZV<4>, 2PZV<35>, 2PZV<35>, 2PZV<1>,
                             ZPZV<32>, ZPZV<16>, ZPZV<1>, ZPZV<9>, ZPZV<2»; }; // NOLINT
    template<> struct ConwayPolynomial<37, 15> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
03646
```

```
ZPZV<28>, ZPZV<27>, ZPZV<13>, ZPZV<34>, ZPZV<33>, ZPZV<35»; };</pre>
                           template<> struct ConwayPolynomial<37, 17> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>; }; // NOLINT
    template<> struct ConwayPolynomial<37, 18> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<8>, ZPZV<19>, ZPZV<15>,
03648
                            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
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<35»; }; //</pre>
                            NOLINT
                                            template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type =
03650
                           POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                               template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
                           POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6»; }; // NOLINT
03652
                                            template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<35»; }; // NOLINT template<> struct ConwayPolynomial<41, 4> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<6»; }; // NOLINT
03653
                                             template<> struct ConwayPolynomial<41, 5> { using ZPZ = aerobus::zpz<41>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<14>, ZPZV<35»; }; // NOLINT
03655
                                            template<> struct ConwayPolynomial<41, 6> { using ZPZ = aerobus::zpz<41>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<33>, ZPZV<39>, ZPZV<6>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<41, 7> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 >, ZPZV<
03656
                                               template<> struct ConwayPolynomial<41, 8> { using ZPZ = aerobus::zpz<41>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<32>, ZPZV<20>, ZPZV<6>, ZPZV<6»; };
03658
                                           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<55>, ZPZV<35»; };
                            // NOLINT
                           template<> struct ConwayPolynomial<41, 10> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<31>, ZPZV<8>, ZPZV<20>, ZPZV<30>,
03659
                            ZPZV<6»; }; // NOLINT
                                           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 =
03661
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<24>, ZPZV<26>, ZPZV<24>, ZPZV<24>, ZPZV<26>, ZPZV<24>, ZPZV<26>, ZPZV<26 ), ZPZV<26 ), ZPZV<27 ), ZPZV<27 ), ZPZV<28 ), ZP
                                            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>,
03663
                            ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35»; }; // NOLINT
    template<> struct ConwayPolynomial<41, 17> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03665
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<41, 18> { using ZPZ = aerobus::zpz<41>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<7>, ZPZV<1>, ZPZV<1
, ZPZV<1
                           ZPZV<23>, ZPZV<35>, ZPZV<38>, ZPZV<24>, ZPZV<12>, ZPZV<29>, ZPZV<10>, ZPZV<6>, ZPZV<6>; }; // NOLINT template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<35»; }; //
03667
                                               template<> struct ConwayPolynomial<43, 1> { using ZPZ = aerobus::zpz<43>; using type =
                          POLYV<ZPZV<1>, ZPZV<40»; }; // NOLINT
                                           template<> struct ConwayPolynomial<43, 2> { using ZPZ = aerobus::zpz<43>; using type =
03669
                         POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3»; }; // NOLINT
03670
                                              template<> struct ConwayPolynomial<43, 3> { using ZPZ = aerobus::zpz<43>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40»; }; // NOLINT template<> struct ConwayPolynomial<43, 4> { using ZPZ = aerobus::zpz<43>; using type =
03671
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3»; }; // NOLINT
                         template<> struct ConwayPolynomial<43, 5> { using ZPZ = aerobus::zpz<43>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<40»; }; // NOLINT
template<> struct ConwayPolynomial<43, 6> { using ZPZ = aerobus::zpz<43>; using type =
03672
03673
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<21>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<43, 7> { using ZPZ = aerobus::zpz<43>; using type =
03674
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<7>, ZPZV<7>, ZPZV<40»; }; // NOLINT
                           template<> struct ConwayPolynomial<43, 8> { using ZPZ = aerobus::zpz<43>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<39>, ZPZV<20>, ZPZV<24>, ZPZV<3»; };</pre>
03675
                           NOLINT
                                              template<> struct ConwayPolynomial<43, 9> { using ZPZ = aerobus::zpz<43>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<39>, ZPZV<1>, ZPZV<40»; };
                                            template<> struct ConwayPolynomial<43, 10> { using ZPZ = aerobus::zpz<43>; using type =
03677
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<26>, ZPZV<36>, ZPZV<36>, ZPZV<5>, ZPZV<5>, ZPZV<27>, ZPZV<24>,
                            ZPZV<3»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<7>, ZPZV<40»; }; // NOLINT</pre>
                                           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<3»; }; // NOLINT</pre>
```

```
template<> struct ConwayPolynomial<43, 13> { using ZPZ = aerobus::zpz<43>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                   template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<38>, ZPZV<22>, ZPZV<24>,
                                 ZPZV<37>, ZPZV<18>, ZPZV<4>, ZPZV<19>, ZPZV<3»; }; // NOLINT</pre>
                                                       template<> struct ConwayPolynomial<43, 15> { using ZPZ = aerobus::zpz<43>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<22>, ZPZV<42>, ZPZV<45>, ZPZV<15>, ZPZV<37>, ZPZV<40»; }; // NOLINT template<> struct ConwayPolynomial
43, 17> { using ZPZ = aerobus::zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZ
03683
                                                      template<> struct ConwayPolynomial<43, 18> { using ZPZ = aerobus::zpz<43>; using type
03684
                                 ZPZV<24>, ZPZV<7>, ZPZV<24>, ZPZV<29>, ZPZV<16>, ZPZV<34>, ZPZV<37>, ZPZV<18>, ZPZV<3»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<43, 19> { using ZPZ = aerobus::zpz<43>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
03685
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>, ZPZV<40»; };</pre>
                                                      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 =
03687
                                POLYV<ZPZV<1>, ZPZV<45>, ZPZV<5»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<47, 3> { using ZPZ = aerobus::zpz<47>; using type =
03688
                                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
03690
                                                     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
                                                    template<> struct ConwayPolynomial<47, 6> { using ZPZ = aerobus::zpz<47>; using type =
03691
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<35>, ZPZV<41>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<47, 7> { using ZPZ = aerobus::zpz<47>; using type
03692
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42»; }; // NOLINT
03693
                                                  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»; }; template<> struct ConwayPolynomial<47, 9> { using ZPZ = aerobus::zpz<47>; using type =
03694
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<42»; };
                                                      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<5»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<47, 11> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
03696
                                template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<40>, ZPZV<40>, ZPZV<12>, ZPZV<46>,
                                 ZPZV<14>, ZPZV<9>, ZPZV<5>; }; // NOLINT
                                                    template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type =
                                POLYY<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<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<30>, ZPZV<30>,
                                  ZPZV<17>, ZPZV<24>, ZPZV<9>, ZPZV<32>, ZPZV<5»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<47, 15> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03700
                                 POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 03702
                                                       template<> struct ConwayPolynomial<47, 18> { using ZPZ = aerobus::zpz<47>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                                                      template<> struct ConwayPolynomial<47, 19> { using ZPZ = aerobus::zpz<47>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>
                                 NOLINT
03704
                                                      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 =
                               POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2»; }; // NOLINT
                                                     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 =
03707
                                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
03708
                                                    template<> struct ConwayPolynomial<53, 6> { using ZPZ = aerobus::zpz<53>; using type =
03709
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<4>, ZPZV<45>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<53, 7> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5), ZPZV<5, ZPZV<5,
03710
                                                      template<> struct ConwayPolynomial<53, 8> { using ZPZ = aerobus::zpz<53>; using type =
03711
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<29>, ZPZV<18>, ZPZV<1>, ZPZV<2»; };
                                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<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>; };
                                 // NOLINT
                                template<> struct ConwayPolynomial<53, 10> { using ZPZ = aerobus::zpz<53>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<27>, ZPZV<27>, ZPZV<15>, ZPZV<29>,
03713
```

```
ZPZV<2»; };</pre>
                                      template<> struct ConwayPolynomial<53, 11> { using ZPZ = aerobus::zpz<53>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<15>, ZPZV<51»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<53, 12> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<34>, ZPZV<4>, ZPZV<13>, ZPZV<10>, ZPZV<42>, ZPZV<34>, ZPZV<41>, ZPZV<41>, ZPZV<2»; }; // NOLINT
                                       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 =
03717
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<1>, ZPZV<45>, ZPZV<45>, ZPZV<23>, ZPZV<52>,
                       ZPZV<0>, ZPZV<37>, ZPZV<12>, ZPZV<23>, ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<53, 15> { using ZPZ = aerobus::zpz<53>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51»; }; // NOLINT
    template<> struct ConwayPolynomial<53, 17> { using ZPZ = aerobus::zpz<53>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                        \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 12>, \ \texttt{ZPZV} < 51»; \ // \ \texttt{NOLINT} 
                                       template<> struct ConwayPolynomial<53, 18> { using ZPZ = aerobus::zpz<53>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<52>, 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>,
                                      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 =
03723
                      POLYY<ZPZV<1>, ZPZV<58>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
03724
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; }; // NOLINT template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
03725
                       03726
                                       template<> struct ConwayPolynomial<59, 5> { using ZPZ = aerobus::zpz<59>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<57»; }; // NOLINT
                                      template<> struct ConwayPolynomial<59, 6> { using ZPZ = aerobus::zpz<59>; using type =
03727
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18>, ZPZV<38>, ZPZV<0>, ZPZV<2»; }; // NOLINT
03728
                                       template<> struct ConwayPolynomial<59, 7> { using ZPZ = aerobus::zpz<59>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<57»; }; // NOLINT
03729
                                    template<> struct ConwayPolynomial<59, 8> { using ZPZ = aerobus::zpz<59>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<32>, ZPZV<2>, ZPZV<50>, ZPZV<2»; };
                       NOLINT
03730
                                     template<> struct ConwayPolynomial<59, 9> { using ZPZ = aerobus::zpz<59>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<47>, ZPZV<57»; };
                       // NOLINT
03731
                                     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
                                       template<> struct ConwayPolynomial<59, 11> { using ZPZ = aerobus::zpz<59>; using type =
03732
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<6>, ZPZV<57»; }; // NOLINT</pre>
03733
                                    template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<51>, ZPZV<21>, ZPZV<38>, ZPZV<8>, ZPZV<1>, ZPZV<2»; }; // NOLINT
03734
                                      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 =
                       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<20>; }; // NOLINT
    template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type =
                       POLYV<ZPZV<1>, 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 =
                       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>; // NOLINT
template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59>; using type =
03738
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<3 , ZPZV<3
                                     template<> struct ConwayPolynomial<59, 19> { using ZPZ = aerobus::zpz<59>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<57»; }; //</pre>
                       NOLINT
                                      template<> struct ConwayPolynomial<61, 1> { using ZPZ = aerobus::zpz<61>; using type =
                       POLYV<ZPZV<1>, ZPZV<59»; }; // NOLINT
                                     template<> struct ConwayPolynomial<61, 2> { using ZPZ = aerobus::zpz<61>; using type =
03741
                       POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; }; // NOLINT
03742
                                     template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; }; // NOLINT template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
03743
                      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
03745
                                    template<> struct ConwayPolynomial<61, 6> { using ZPZ = aerobus::zpz<61>; using type =
                    POLYV<2PZV<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 =
03746
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; };
                                                  template<> struct ConwayPolynomial<61, 8> { using ZPZ = aerobus::zpz<61>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<1>, ZPZV<56>, ZPZV<2°, ; };
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               // NOLINT
03748
                                              template<> struct ConwayPolynomial<61, 9> { using ZPZ = aerobus::zpz<61>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<50>, ZPZV<50>, ZPZV<59»; };
                              // 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>,
                               ZPZV<2»; }; // NOLINT</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>
                                                  template<> struct ConwayPolynomial<61, 12> { using ZPZ = aerobus::zpz<61>; using type =
03751
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<42>, ZPZV<33>, ZPZV<38>, ZPZV<38>, ZPZV<14>, ZPZV<15>, ZPZV<2»; }; // NOLINT
03752
                                                template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                   template<> struct ConwayPolynomial<61, 14> { using ZPZ = aerobus::zpz<61>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<48>, ZPZV<46>, ZPZV<11>,
                              ZPZV<8>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<2»; }; // NOLINT</pre>
03754
                                               template<> struct ConwayPolynomial<61, 15> { using ZPZ = aerobus::zpz<61>; using type =
                            template<> struct ConwayPolynomial<61, 15> { using ZYZ = aerobus::zpz<61>; using Cype - PoLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<3>, ZPZV<35>, ZPZV<44>, ZPZV<25>, ZPZV<23>, ZPZV<21>, ZPZV<51>, ZPZV<59>; }; // NOLINT template<> struct ConwayPolynomial<61, 17> { using ZPZ = aerobus::zpz<61>; using type = PoLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZP
03755
                              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<42, ZPZV<42, ZPZV<25, ZPZV<56, ZPZV<56, ZPZV<61, ZPZV<28, ZPZV<28, ZPZV<57>, ZPZV<42, ZPZV<25, ZPZV<56, ZPZV<57, ZPZV<
                              POLYV<2PZV<1>, 2PZV<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<59»; }; //</pre>
03758
                                                 template<> struct ConwayPolynomial<67, 1> { using ZPZ = aerobus::zpz<67>; using type =
                            POLYV<ZPZV<1>, ZPZV<65»; }; // NOLINT
03759
                                                template<> struct ConwayPolynomial<67, 2> { using ZPZ = aerobus::zpz<67>; using type =
                            POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
                             POLYV<ZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type =
03761
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<54>, ZPZV<2»; }; // NOLINT

template<> struct ConwayPolynomial<67, 5> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<65»; }; // NOLINT
03762
                                                  template<> struct ConwayPolynomial<67, 6> { using ZPZ = aerobus::zpz<67>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<63>, ZPZV<49>, ZPZV<55>, ZPZV<2»; }; // NOLINT
03764
                                              template<> struct ConwayPolynomial<67, 7> { using ZPZ = aerobus::zpz<67>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<65»; }; // NOLINT
                             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<2»; };
03765
                                                 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<45>, ZPZV<65»; };
                               // NOLINT
                             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>,
03767
                              ZPZV<2»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<67, 11> { using ZPZ = aerobus::zpz<67>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<9>, ZPZV<65»; }; // NOLINT
   template<> struct ConwayPolynomial<67, 12> { using ZPZ = aerobus::zpz<67>; using type =
03769
                             POLYYCZPZV<1>, ZPZV<2), ZPZV<2), ZPZV<3>, ZPZV<3>, ZPZV<5>, ZPZV<4>, ZPZV<64>, ZPZV<64>, ZPZV<64>, ZPZV<55, ZPZV<64>, ZPZV<65, ZPZV<64>, ZPZV<65, ZPZV<64>, ZPZV<64>, ZPZV<65, ZPZV<64>, ZPZV<64>, ZPZV<65, ZPZV<64>, ZPZV<65, ZPZV<64>, ZPZV<65, ZPZV<64>, ZPZV<64>, ZPZV<64>, ZPZV<64>, ZPZV<64>, ZPZV<65, ZPZV<64, ZPZV<65, ZPZV<65,
                                                  template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<65»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<67, 14> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<22>, ZPZV<5>,
ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<3>, ZPZV<2»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<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>
03773
                             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
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<63>, ZPZV<63>, ZPZV<63>, ZPZV<52>, ZPZV<18>,
                               ZPZV<33>, ZPZV<55>, ZPZV<28>, ZPZV<29>, ZPZV<51>, ZPZV<6>, ZPZV<59>, ZPZV<13>, ZPZV<2»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<67, 19> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                                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 =
03777
                           POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; };
                           template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, ZPZV<7»; }; // NOLINT
                                               template<> struct ConwayPolynomial<71, 5> { using ZPZ = aerobus::zpz<71>; using type =
 03780
                             \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<18>, \verb"ZPZV<64"; \verb"]; $ // \verb"NOLINT" | NOLINT" | NOLINT"
03781
                                                template<> struct ConwayPolynomial<71, 6> { using ZPZ = aerobus::zpz<71>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<10>, ZPZV<29>, ZPZ
 03782
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<64*; }; // NOLINT
03783
                                             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
03784
                                               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<62>, ZPZV<64»; };
                             // NOLINT
03785
                                               template<> struct ConwayPolynomial<71, 10> { using ZPZ = aerobus::zpz<71>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17>, ZPZV<26>, ZPZV<26>, ZPZV<40>,
                             ZPZV<7»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<71, 11> { using ZPZ = aerobus::zpz<71>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                             ZPZV<48>, ZPZV<64»; }; // NOLINT</pre>
03787
                                              template<> struct ConwayPolynomial<71, 12> { using ZPZ = aerobus::zpz<71>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<28>, ZPZV<29>, ZPZV<25>, ZPZV<21>, ZPZV<58>, ZPZV<23>, ZPZV<3>, ZPZV<3
03788
                                               template<> struct ConwayPolynomial<71, 13> { using ZPZ = aerobus::zpz<71>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              template<> struct ConwayPolynomial<71, 15> { using ZPZ = aerobus::zpz<71>; using type :
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<28>, ZPZV<32>, ZPZV<18>, ZPZV<52>, ZPZV<67>, ZPZV<49>, ZPZV<64»; }; // NOLINT template<> struct ConwayPolynomial<71, 17> { using ZPZ = aerobus::zpz<71>; using type =
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                              template<> struct ConwayPolynomial<71, 19> { using ZPZ = aerobus::zpz<71>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<6>, ZPZV<6 , ZPZV<6
                             NOLINT
                                                template<> struct ConwayPolynomial<73, 1> { using ZPZ = aerobus::zpz<73>; using type =
                             POLYV<ZPZV<1>, ZPZV<68»; }; // NOLINT
                                              template<> struct ConwayPolynomial<73, 2> { using ZPZ = aerobus::zpz<73>; using type =
                          POLYV<2PZV<1>, ZPZV<70>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
03794
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68»; }; // NOLINT template<> struct ConwayPolynomial<73, 4> { using ZPZ = aerobus::zpz<73>; using type =
 03795
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5»; }; // NOLINT
 03796
                                               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
 03797
                                              template<> struct ConwayPolynomial<73, 6> { using ZPZ = aerobus::zpz<73>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<45>, ZPZV<23>, ZPZV<48>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<73, 7> { using ZPZ = aerobus::zpz<73>; using type =
 03798
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<68»; }; // NOLINT
                                               template<> struct ConwayPolynomial<73, 8> { using ZPZ = aerobus::zpz<73>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<53>, ZPZV<39>, ZPZV<18>, ZPZV<5»; };
03800
                                              template<> struct ConwayPolynomial<73, 9> { using ZPZ = aerobus::zpz<73>; using type =
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<72>, ZPZV<15>, ZPZV<68»; };
                                               template<> struct ConwayPolynomial<73, 10> { using ZPZ = aerobus::zpz<73>; using type =
 03801
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<23>, ZPZV<33>, ZPZV<33>, ZPZV<36>, ZPZV<69>,
                             ZPZV<5»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<73, 11> { using ZPZ = aerobus::zpz<73>; using type =
03802
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<5>, ZPZV<68»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<73, 12> { using ZPZ = aerobus::zpz<73>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<52>, ZPZV<26>, ZPZV<20>, ZPZV<46>,
                             ZPZV<29>, ZPZV<25>, ZPZV<5»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<73, 13> { using ZPZ = aerobus::zpz<73>; using type
03804
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                 template<> struct ConwayPolynomial<73, 15> { using ZPZ = aerobus::zpz<73>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             template<> struct ConwayPolynomial<73, 17> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03806
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<88, ZPZV<88; }; // NOLINT
template<> struct ConwayPolynomial<73, 19> { 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>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<68»; }; //</pre>
                             NOLINT
03808
                                               template<> struct ConwayPolynomial<79, 1> { using ZPZ = aerobus::zpz<79>; using type =
                            POLYV<ZPZV<1>, ZPZV<76»; }; // NOLINT
                                                template<> struct ConwayPolynomial<79, 2> { using ZPZ = aerobus::zpz<79>; using type =
                            POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3»; }; // NOLINT
 03810
                                             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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3»; }; // NOLINT
 03811
```

```
template<> struct ConwayPolynomial<79, 5> { using ZPZ = aerobus::zpz<79>; using type =
                       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<1>, ZPZV<2>, ZPZV<28>, ZPZV<68>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<79, 7> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<6»; }; // NOLINT
03814
                                      template<> struct ConwayPolynomial<79, 8> { using ZPZ = aerobus::zpz<79>; using type =
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<60>, ZPZV<59>, ZPZV<48>, ZPZV<3»; };
                       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<76»; };</pre>
03816
                        // NOLINT
03817
                                       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>
03818
                                      template<> struct ConwayPolynomial<79, 11> { using ZPZ = aerobus::zpz<79>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        template<> struct ConwayPolynomial<79, 12> { using ZPZ = aerobus::zpz<79>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<45>, ZPZV<52>, ZPZV<52>, ZPZV<40>,
                        ZPZV<59>, ZPZV<62>, ZPZV<3»; }; // NOLINT</pre>
03820
                                     template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<78>, ZPZV<76»; }; // NOLINT template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type =
03821
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<76»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<79, 19> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03822
                        NOLINT
03823
                                        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 =
03824
                       POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; }; // NOLINT
03825
                                       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 =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2»; }; // NOLINT
                                     template<> struct ConwayPolynomial<83, 5> { using ZPZ = aerobus::zpz<83>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<81»; }; // NOLINT
                                      template<> struct ConwayPolynomial<83, 6> { using ZPZ = aerobus::zpz<83>; using type =
03828
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<76>, ZPZV<32>, ZPZV<17>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<83, 7> { using ZPZ = aerobus::zpz<83>; using type
03829
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT
03830
                                      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<42»; }; //
                       NOLINT
03831
                       template<> struct ConwayPolynomial<83, 9> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<281»; };</pre>
                        // NOLINT
                                       template<> struct ConwayPolynomial<83, 10> { using ZPZ = aerobus::zpz<83>; using type =
03832
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5 , ZPZV<5
                        ZPZV<2»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type =
03833
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type =
03834
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35>, ZPZV<31>, ZPZV<31>, ZPZV<65>, ZPZV<55>, ZPZV<75>, ZPZV<2»; }; // NOLINT
                                      template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type =
03835
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<81»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<83, 17> { using ZPZ = aerobus::zpz<83>; using type
                       POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<81»; }; // NOLINT</pre>
03837
                       template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<81»; }; //</pre>
                        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 =
03839
                       POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; }; // NOLINT
                                      template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
03840
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<86»; }; // NOLINT
                                     template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
03841
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<89, 5> { using ZPZ = aerobus::zpz<89>; using type =
03842
                     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 =
03843
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<82>, ZPZV<80>, ZPZV<15>, ZPZV<3»; }; // NOLINT
                                       template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<86»; };
03845
                                    template<> struct ConwayPolynomial<89, 8> { using ZPZ = aerobus::zpz<89>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<65>, ZPZV<40>, ZPZV<79>, ZPZV<3»; }; //
```

```
template<> struct ConwayPolynomial<89, 9> { using ZPZ = aerobus::zpz<89>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<12>, ZPZV<6>, ZPZV<86»; };
                          // NOLINT
                          template<> struct ConwayPolynomial<89, 10> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<33>, ZPZV<82>, ZPZV<52>, ZPZV<4>,
03847
                          ZPZV<3»: }: // NOLINT
                                            template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<26>, ZPZV<86»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<89, 12> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<85>, ZPZV<15>, ZPZV<44>, ZPZV<51>, ZPZV<8>,
ZPZV<70>, ZPZV<52>, ZPZV<3»; }; // NOLINT</pre>
03849
                                            template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type
03850
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            template<> struct ConwayPolynomial<89, 17> { using ZPZ = aerobus::zpz<89>; using type =
03851
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6*, ZPZV<6*); // NOLINT template<> struct ConwayPolynomial<89, 19> { using ZPZ = aerobus::zpz<89>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0</pre>
03853
                                            template<> struct ConwayPolynomial<97, 1> { using ZPZ = aerobus::zpz<97>; using type =
                          POLYV<ZPZV<1>, ZPZV<92»; }; // NOLINT
03854
                                             template<> struct ConwayPolynomial<97, 2> { using ZPZ = aerobus::zpz<97>; using type =
                          POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5»; }; // NOLINT
                                          template<> struct ConwayPolynomial<97, 3> { using ZPZ = aerobus::zpz<97>; using type =
03855
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<92»; }; // NOLINT template<> struct ConwayPolynomial<97, 4> { using ZPZ = aerobus::zpz<97>; using type =
03856
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<97, 5> { using ZPZ = aerobus::zpz<97>; using type =
03857
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<92»; }; // NOLINT
                                            template<> struct ConwayPolynomial<97, 6> { using ZPZ = aerobus::zpz<97>; using type =
03858
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<92>, ZPZV<58>, ZPZV<88>, ZPZV<5»; }; // NOLINT
                         template<> struct ConwayPolynomial<97, 7> { using ZPZ = aerobus::2pz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<92»; }; // NOLINT</pre>
03859
                                          template<> struct ConwayPolynomial<97, 8> { using ZPZ = aerobus::zpz<97>; using type =
03860
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<1>, ZPZV<32>, ZPZV<5»; };
                                             template<> struct ConwayPolynomial<97, 9> { using ZPZ = aerobus::zpz<97>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59>, ZPZV<7>, ZPZV<92»; };
                          // NOLINT
                          template<> struct ConwayPolynomial<97, 10> \{ using ZPZ = \frac{aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<66>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<20>,
03862
                          ZPZV<5»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<5>, ZPZV<92»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<97, 12> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<59>, ZPZV<81>, ZPZV<86>, ZPZV<78>, ZPZV<94>, ZPZV<59; }; // NOLINT
03865
                                            template<> struct ConwayPolynomial<97, 13> { using ZPZ = aerobus::zpz<97>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<92»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomials97, 17> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03866
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<52, ZPZV<92w; }; // NOLINT
template<> struct ConwayPolynomial<97, 19> { using ZPZ = aerobus::zpz<97>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<15>, ZPZV<92»; }; //</pre>
                          NOLINT
03868
                                          template<> struct ConwayPolynomial<101, 1> { using ZPZ = aerobus::zpz<101>; using type =
                          POLYV<ZPZV<1>, ZPZV<99»; }; // NOLINT
03869
                                             template<> struct ConwayPolynomial<101, 2> { using ZPZ = aerobus::zpz<101>; using type =
                          POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; }; // NOLINT
03870
                                          template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<9>»; }; // NOLINT
template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<78>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<101, 5> { using ZPZ = aerobus::zpz<101>; using type =
03871
03872
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<99»; }; // NOLINT
                                            template<> struct ConwayPolynomial<101, 6> { using ZPZ = aerobus::zpz<101>; using type =
03873
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<90>, ZPZV<20>, ZPZV<67>, ZPZV<2*; }; // NOLINT
                                            template<> struct ConwayPolynomial<101, 7> { using ZPZ = aerobus::zpz<101>; using type =
03874
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<9»; }; // NOLINT
                                            template<> struct ConwayPolynomial<101, 8> { using ZPZ = aerobus::zpz<101>; using type =
03875
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76>, ZPZV<29>, ZPZV<24>, ZPZV<2*; };
03876
                                          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<6>, ZPZV<64>, ZPZV<47>, ZPZV<499»; };
                           // NOLINT
                                           template<> struct ConwayPolynomial<101, 10> { using ZPZ = aerobus::zpz<101>; using type =
03877
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<67>, ZPZV<49>, ZPZV<100>, ZPZV<100>, ZPZV<52>,
                          ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<101, 11> { using ZPZ = aerobus::zpz<101>; using type :
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03879
                                         template<> struct ConwayPolynomial<101, 12> { using ZPZ = aerobus::zpz<101>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<79>, ZPZV<64>, ZPZV<39>, ZPZV<78>, ZPZV<48>,
                            ZPZV<84>, ZPZV<21>, ZPZV<2»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<101, 13> { using ZPZ = aerobus::zpz<101>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<99»; };
                                                                                                                                                                                                                           // NOLINT
                                              template<> struct ConwayPolynomial<101, 17> { using ZPZ = aerobus::zpz<101>; using type =
03881
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<99»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                                                      // NOLINT
                                            template<> struct ConwayPolynomial<101, 19> { using ZPZ = aerobus::zpz<101>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2+, ZPZV<9*; }; //</pre>
                            NOLINT
03883
                                              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 =
03885
                           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<20, ZPZY<2>, ZPZY<20, ZPZY<2
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<88>, ZPZV<5»; }; // NOLINT
                                               template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<98»; }; // NOLINT
                                             template<> struct ConwayPolynomial<103, 6> { using ZPZ = aerobus::zpz<103>; using type =
03888
                          POLYVCZPZVC1>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<96>, ZPZV<30>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<103, 7> { using ZPZ = aerobus::2pz<103>; using type =
03889
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<9»; }; // NOLINT
                                             template<> struct ConwayPolynomial<103, 8> { using ZPZ = aerobus::zpz<103>; using type
03890
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<70>, ZPZV<71>, ZPZV<49>, ZPZV<5»; };
                            NOLINT
03891
                                             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<51>, ZPZV<51
                             // NOLINT
03892
                                              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>
03893
                                              template<> struct ConwayPolynomial<103, 11> { using ZPZ = aerobus::zpz<103>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                               template<> struct ConwayPolynomial<103, 12> { using ZPZ = aerobus::zpz<103>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<74>, ZPZV<23>, ZPZV<94>, ZPZV<20>, ZPZV<81>, ZPZV<88>, ZPZV<88>, ZPZV<88>, ZPZV<5»; }; // NOLINT
03895
                                             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
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            template<> struct ConwayPolynomial<103, 19> { using ZPZ = aerobus::zpz<103>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03897
                            NOLINT
                                               template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
                           POLYV<ZPZV<1>, ZPZV<105»; }; // NOLINT template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
03899
                          POLYV<ZPZV<1>, ZPZV<103>, ZPZV<2»; }; // NOLINT
                                              template<> struct ConwayPolynomial<107, 3> { using ZPZ = aerobus::zpz<107>; using type =
03900
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<105»; }; // NOLINT
                                            template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
03901
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<13>, ZPZV<29>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<107, 5> { using ZPZ = aerobus::zpz<107>; using type =
03902
                          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 =
03903
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<52>, ZPZV<22>, ZPZV<79>, ZPZV<2»; }; // NOLINT
                                              template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<16>, ZPZV<165»; };
03905
                                           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>, ZPZV<2»; }; //
                           NOLINT
                                             template<> struct ConwayPolynomial<107, 9> { using ZPZ = aerobus::zpz<107>; using type =
03906
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<66>, ZPZV<105»; };
03907
                                           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 =
03908
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                          // NOLINT
                            ZPZV<8>, ZPZV<105»; };</pre>
03909
                                             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<48>, ZPZV<6>, ZPZV<6>, ZPZV<61>,
                            ZPZV<42>, ZPZV<57>, ZPZV<2»: 1: // NOLINT
                                               template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type :
03910
                             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<4>, ZPZV<105»; };</pre>
                                                                                                                                                                                                                         // NOLINT
03911
                                              template<> struct ConwayPolynomial<107, 17> { using ZPZ = aerobus::zpz<107>; using type :
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
03912
```

```
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<24>, ZPZV<105»; }; //</pre>
                                  NOLINT
03913
                                                      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 =
03914
                                 POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; }; // NOLINT
                                                         template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
 03915
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<13»; }; // 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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<0>, ZPZV<10>, ZPZV<0>, ZPZV<0 , Z
 03916
03917
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103»; }; // NOLINT
                                                         template<> struct ConwayPolynomial<109, 6> { using ZPZ = aerobus::zpz<109>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<102>, ZPZV<66>, ZPZV<6%; }; // NOLINT
 03919
                                                      template<> struct ConwayPolynomial<109, 7> { using ZPZ = aerobus::zpz<109>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<14>, ZPZV<103»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<109, 8> { using ZPZ = aerobus::zpz<109>; using type =
03920
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<34>, ZPZV<86>, ZPZV<6»; };
                                 template<> struct ConwayPolynomial<109, 9> { using ZPZ = aerobus::zpz<109>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
                                   // NOLINT
                                                      template<> struct ConwayPolynomial<109, 10> { using ZPZ = aerobus::zpz<109>; using type =
03922
                                  POLYY<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 =
03923
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<11>, ZPZV<103»; }; // NOLINT</pre>
                                                      template<> struct ConwayPolynomial<109, 12> { using ZPZ = aerobus::zpz<109>; using type =
03924
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<55>, ZPZV<53>, ZPZV<37>, ZPZV<85>, ZPZV<65>,
                                  ZPZV<103>, ZPZV<28>, ZPZV<6»; }; // NOLINT</pre>
                                                        template<> struct ConwayPolynomial<109, 13> { using ZPZ = aerobus::zpz<109>; using type
03925
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT
    template<> struct ConwayPolynomial<109, 17> { using ZPZ = aerobus::zpz<109>; using type =
03926
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03927
                                                        template<> struct ConwayPolynomial<109, 19> { using ZPZ = aerobus::zpz<109>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<15>, ZPZV<103»; }; //</pre>
                                  NOLINT
                                                       template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
03928
                                 POLYV<ZPZV<1>, ZPZV<110»; }; // NOLINT
                                                         template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
                                 POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3»; }; // NOLINT
 03930
                                                     template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<110»; }; // NOLINT template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
03931
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<62>, ZPZV<3»; }; // NOLINT
                                                        template<> struct ConwayPolynomial<113, 5> { using ZPZ = aerobus::zpz<113>; using type =
03932
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110»; }; // NOLINT
 03933
                                                      template<> struct ConwayPolynomial<113, 6> { using ZPZ = aerobus::zpz<113>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5), ZPZV<5), ZPZV<5), ZPZV<30>, ZPZV<30; }; // NOLINT template<> struct ConwayPolynomial<113, 7> { using ZPZ = aerobus::zpz<113>; using type =
03934
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZP
                                  POLYV<PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<38>, ZPZV<38>, ZPZV<28>, ZPZV<28>, ZPZV<3»; };
03936
                                                        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<1>, ZPZV<1>, ZPZV<1>, ZPZV<10, ZPZV<1
                                  // NOLINT
03937
                                                        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>
03938
                                                    template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<3>, ZPZV<110»: }: // NOLINT</pre>
                                                      template<> struct ConwayPolynomial<113, 12> { using ZPZ = aerobus::zpz<113>; using type =
                                 POLYY<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<5>, ZPZV<5 , ZPZV<5
03940
                                                    template<> struct ConwayPolynomial<113, 13> { using ZPZ = aerobus::zpz<113>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                        template<> struct ConwayPolynomial<113, 17> { using ZPZ = aerobus::zpz<113>; using type =
03941
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110»; };</pre>
03942
                                                      template<> struct ConwayPolynomial<113, 19> { using ZPZ = aerobus::zpz<113>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  NOLINT
                                                        template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
                                 POLYV<ZPZV<1>, ZPZV<124»; }; // NOLINT
 03944
                                                     template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
                                POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124»; }; // NOLINT
 03945
```

```
03946
                                        template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type =
                      POLYV<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
                      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<3»; }; // NOLINT</pre>
 03948
                                       template<> struct ConwayPolynomial<127, 7> { using ZPZ = aerobus::zpz<127>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<124»; };
                                    template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type =
 03950
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<104>, ZPZV<55>, ZPZV<8>, ZPZV<8»; };
                       NOLINT
                                     template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type =
03951
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12+, ZPZV<14>, ZPZV<119>, ZPZV<126>, ZPZV<124»;
                       }; // NOLINT
                                    template<> struct ConwayPolynomial<127, 10> { using ZPZ = aerobus::zpz<127>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<64>, ZPZV<95>, ZPZV<60>, ZPZV<4+,
                       ZPZV<3»; }; // NOLTNT</pre>
                                     template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type
03953
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<11>, ZPZV<124»; };</pre>
                                                                                                                   // NOLINT
                       template<> struct ConwayPolynomial<127, 12> { using ZPZ = aerobus::zpz<127>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<33>, ZPZV<33>, ZPZV<97>, ZPZV<15>,
                       ZPZV<99>, ZPZV<8>, ZPZV<3»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type =
03955
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<124»; };</pre>
                                                                                                                                                                                      // NOLINT
                                    template<> struct ConwayPolynomial<127, 17> { using ZPZ = aerobus::zpz<127>; using type =
03956
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                                                                                                                                                                                                                                                                                                             // NOLINT
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<124»; };</pre>
                                     template<> struct ConwayPolynomial<127, 19> { using ZPZ = aerobus::zpz<127>; using type =
03957
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30</pre>
                                     template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
03958
                       POLYV<ZPZV<1>, ZPZV<129»; }; // NOLINT
                                       template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
03959
                       POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2»; }; // NOLINT
                                       template<> struct ConwayPolynomial<131, 3> { using ZPZ = aerobus::zpz<131>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<129»; }; // NOLINT template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type =
 03961
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2»; }; // NOLINT
                       template<> struct ConwayPolynomial<ali>131, 5> { using ZPZ = aerobus::zpz<131>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<129»; }; // NOLINT
03962
03963
                                      template<> struct ConwayPolynomial<131, 6> { using ZPZ = aerobus::zpz<131>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<4>, ZPZV<22>, ZPZV<2»; }; // NOLINT
 03964
                                       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»; };
                                                                                                                                                                                                                                                                                                                                                         // NOLINT
                                    template<> struct ConwayPolynomial<131, 8> { using ZPZ = aerobus::zpz<131>; using type =
 03965
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<72>, ZPZV<116>, ZPZV<104>, ZPZV<2»; };
                       NOLINT
03966
                                      template<> struct ConwayPolynomial<131, 9> { using ZPZ = aerobus::zpz<131>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<19>, ZPZV<129»; };
                        // NOLINT
03967
                                       template<> struct ConwayPolynomial<131, 10> { using ZPZ = aerobus::zpz<131>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<124>, ZPZV<97>, ZPZV<9>, ZPZV<126>, ZPZV<44>,
                       ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<6>, ZPZV<129»; }; // NOLINT</pre>
03969
                                       template<> struct ConwayPolynomial<131, 12> { using ZPZ = aerobus::zpz<131>; using type :
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<50>, ZPZV<40>, ZPZV<83>, ZPZV<125>, ZPZV<28>, ZPZV<203>, ZPZV<28; }; // NOLINT
03970
                                       template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                      // NOLINT
                       ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<129»; };</pre>
03971
                                    template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
                        \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<0>; ZPZV<0>, ZPZ
03972
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<129»; }; //</pre>
03973
                                      template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
                       POLYV<ZPZV<1>, ZPZV<134»; }; // NOLINT
                                       template<> struct ConwayPolynomial<137, 2> { using ZPZ = aerobus::zpz<137>; using type =
03974
                       POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3»; }; // NOLINT
                                     template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type =
 03975
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
 03976
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
 03977
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<134»; }; // NOLINT
                                       template<> struct ConwayPolynomial<137, 6> { using ZPZ = aerobus::zpz<137>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<116>, ZPZV<102>, ZPZV<3>, ZPZV<3»; }; // NOLINT
 03979
                                    template<> struct ConwayPolynomial<137, 7> { using ZPZ = aerobus::zpz<137>; using type =
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 8> { using ZPZ = aerobus::zpz<137>; using type =
 03980
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105>, ZPZV<21>, ZPZV<34>, ZPZV<3*; }; //
03981
                             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<1>, ZPZV<122>, ZPZV<134»;
                  }; // NOLINT
                               template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type =
03982
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<20>, ZPZV<67>, ZPZV<67>, ZPZV<93>, ZPZV<119>,
                  ZPZV<3»; }; // NOLINT
                             template<> struct ConwayPolynomial<137, 11> { using ZPZ = aerobus::zpz<137>; using type =
03983
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  ZPZV<1>, ZPZV<134»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<137, 12> { using ZPZ = aerobus::zpz<137>; using type
03984
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<61>, ZPZV<40>, ZPZV<40>, ZPZV<36>, ZPZV<36>, ZPZV<135>, ZPZV<61>, ZPZV<61>, ZPZV<3»; }; // NOLINT
03985
                            template<> struct ConwayPolynomial<137, 13> { using ZPZ = aerobus::zpz<137>; using type
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 17> { using ZPZ = aerobus::zpz<137>; using type =
                                                                                                                                                         17> { using ZPZ = aerobus::zpz<137>; using type =
03986
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<4>, ZPZV<134»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<137, 19> { using ZPZ = aerobus::zpz<137>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  ZPZV<0>, ZPZV<18>, ZPZV<14*; }; //</pre>
                  NOLINT
03988
                               template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
                  POLYV<ZPZV<1>, ZPZV<137»; }; // NOLINT
                             template<> struct ConwayPolynomial<139, 2> { using ZPZ = aerobus::zpz<139>; using type =
03989
                  POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2»; }; // NOLINT
03990
                               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 =
03991
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2»; };
                                                                                                                                                                                              // NOLINT
                               template<> struct ConwayPolynomial<139, 5> { using ZPZ = aerobus::zpz<139>; using type =
03992
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<137»; }; // NOLINT
03993
                               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
                             template<> struct ConwayPolynomial<139, 7> { using ZPZ = aerobus::zpz<139>; using type =
03994
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<137»; }; // NOLINT
                               template<> struct ConwayPolynomial<139, 8> { using ZPZ = aerobus::zpz<139>; using type
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>, ZPZV<36>, ZPZV<21>, ZPZV<2»; };
                 template<> struct ConwayPolynomial<139, 9> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<70>, ZPZV<70>, ZPZV<87>, ZPZV<137»; };</pre>
03996
                  // NOLINT
                  template<> struct ConwayPolynomial<139, 10> { using ZPZ = aerobus::zpz<139>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<110>, ZPZV<48>, ZPZV<130>, ZPZV<66>,
                  ZPZV<106>, ZPZV<2»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<139, 11> { using ZPZ = aerobus::zpz<139>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<137»; }; // NOLINT
                               template<> struct ConwayPolynomial<139, 12> { using ZPZ = aerobus::zpz<139>; using type
03999
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<120>, ZPZV<75>, ZPZV<41>, ZPZV<41, ZPZV<77, ZPZV<106>,
                   ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
04000
                               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</pre>
                               template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type
                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137»; };</pre>
                                                                                                                                                                                                                                                                  // NOLINT
04002
                               template<> struct ConwayPolynomial<139, 19> { using ZPZ = aerobus::zpz<139>; using type =
                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                  NOLINT
                               template<> struct ConwayPolynomial<149, 1> { using ZPZ = aerobus::zpz<149>; using type =
                  POLYV<ZPZV<1>, ZPZV<147»; }; // NOLINT
04004
                             template<> struct ConwayPolynomial<149, 2> { using ZPZ = aerobus::zpz<149>; using type =
                  POLYV<ZPZV<1>, ZPZV<145>, ZPZV<2»; }; // NOLINT
                              template<> struct ConwayPolynomial<149, 3> { using ZPZ = aerobus::zpz<149>; using type =
04005
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147»; }; // NOLINT
                               template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
04006
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<149, 5> { using ZPZ = aerobus::zpz<149>; using type =
04007
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 6> { using ZPZ = aerobus::zpz<149>; using type =
04008
                  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 =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<19>, ZPZV<147»; };
04010
                             template<> struct ConwayPolynomial<149, 8> { using ZPZ = aerobus::zpz<149>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<140>, ZPZV<25>, ZPZV<123>, ZPZV<2»; }; //
                  NOLINT
                               template<> struct ConwayPolynomial<149, 9> { using ZPZ = aerobus::zpz<149>; using type
04011
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<146>, ZPZV<20, ZPZV<147»;
                  }; // NOLINT
04012
                             template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<74>, ZPZV<42>, ZPZV<148>, ZPZV<143>, ZPZV<51>, ZPZV<2»; }; // NOLINT
04013
                            template<> struct ConwayPolynomial<149, 11> { using ZPZ = aerobus::zpz<149>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                ZPZV<33>, ZPZV<147»; }; // NOLINT</pre>
04014
                                                  template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<91>, ZPZV<52>, ZPZV<9>,
                                ZPZV<104>, ZPZV<110>, ZPZV<2»; }; // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<149, 13> { using ZPZ = aerobus::zpz<149>; using type =
04015
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<147»; };</pre>
                                                                                                                                                                                                                                                            // NOLINT
                                                  template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type =
04016
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<147»; }; // NOLINT
template<> struct ConwayPolynomial<149, 19> { using ZPZ = aerobus::zpz<149>; using type =
                                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<147»; }; //</pre>
                                NOLINT
04018
                                                     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 =
04019
                                POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<151, 3> { using ZPZ = aerobus::zpz<151>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<145»; }; // NOLINT template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
04021
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<151, 5> { using ZPZ = aerobus::zpz<151>; using type =
04022
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<145»; }; // NOLINT
                                                      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
04024
                                                   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<5, Z
04025
                                                   template<> struct ConwayPolynomial<151, 8> { using ZPZ = aerobus::zpz<151>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<140>, ZPZV<122>, ZPZV<43>, ZPZV<6»; }; //
                               NOLINT
                                                    \texttt{template<> struct ConwayPolynomial<151, 9> \{ using \ ZPZ = aerobus:: zpz<151>; \ using \ type = 1000 \ arcsec = 1000 \ arc
04026
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<125>, ZPZV<96>, ZPZV<96 , ZPZV<97 ,
                                }; // NOLINT
04027
                                                     template<> struct ConwayPolynomial<151, 10> { using ZPZ = aerobus::zpz<151>; using type :
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<21>, ZPZV<104>, ZPZV<49>, ZPZV<20>, ZPZV<142>,
                                ZPZV<6»; }; // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<151, 11> { using ZPZ = aerobus::zpz<151>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<145»; }; // NOLINT
                               template<> struct ConwayPolynomial<151, 12> { using ZPZ = aerobus::zpz<151>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<121>, ZPZV<101>, ZPZV<6>, ZPZV<7>>,
04029
                                ZPZV<107>, ZPZV<147>, ZPZV<6»; }; // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<151, 13> { using ZPZ = aerobus::zpz<151>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<145»; }; // NOLINT</pre>
04031
                                                     template<> struct ConwayPolynomial<151, 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<24>, ZPZV<145»; }; // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<151, 19> { using ZPZ = aerobus::zpz<151>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0</pre>, ZPZV<0>, ZP
                                NOLINT
                                                    \texttt{template} <> \texttt{struct ConwayPolynomial} < 157, 1 > \{ \texttt{using ZPZ = aerobus:: zpz} < 157 >; \texttt{using type = aerobus:: zpz} < 157 >; using type = aerobus:: zpz < 157 >; usin
04033
                               POLYV<ZPZV<1>, ZPZV<152»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
                               POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<157, 3> { using ZPZ = aerobus::zpz<157>; using type =
04035
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<152»; }; // NOLINT template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
04036
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<136, ZPZV<38*; }; // NOLINT template<> struct ConwayPolynomial<157, 5> { using ZPZ = aerobus::zpz<157>; using type =
04037
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<152»; }; // NOLINT
04038
                                                  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 =
04039
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<157, 8> { using ZPZ = aerobus::zpz<157>; using type =
                                POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<97>, ZPZV<40>, ZPZV<153>, ZPZV<5»; };
04041
                                                 template<> struct ConwayPolynomial<157, 9> { using ZPZ = aerobus::zpz<157>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<114>, ZPZV<52>, ZPZV<152»;
                                }; // NOLINT
                                                      template<> struct ConwayPolynomial<157, 10> { using ZPZ = aerobus::zpz<157>; using type =
04042
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<61>, ZPZV<22>, ZPZV<124>, ZPZV<61>, ZPZV<93>,
                                ZPZV<5»; }; // NOLINT</pre>
04043
                                                   template<> struct ConwayPolynomial<157, 11> { using ZPZ = aerobus::zpz<157>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<29>, ZPZV<152»; }; // NOLINT
   template<> struct ConwayPolynomial<157, 12> { using ZPZ = aerobus::zpz<157>; using type =
04044
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<71>, ZPZV<110>, ZPZV<72>, ZPZV<137>, ZPZV<43>,
                                ZPZV<152>, ZPZV<57>, ZPZV<5»; }; // NOLINT</pre>
04045
                                                     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 =
04046
```

```
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<12>, ZPZV<152»; }; // NOLINT
template<> struct ConwayPolynomial<157, 19> { using ZPZ = aerobus::zpz<157>; using type =
04047
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; }; //</pre>
                         NOLINT
                                          template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
                         POLYV<ZPZV<1>, ZPZV<161»; }; // NOLINT
                                       template<> struct ConwayPolynomial<163, 2> { using ZPZ = aerobus::zpz<163>; using type =
                       POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
04050
                         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<7>, ZPZY<161»; }; // NOLINT template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
04051
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2»; }; // NOLINT
 04052
                                       template<> struct ConwayPolynomial<163, 5> { using ZPZ = aerobus::zpz<163>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<3>, ZPZV<161»; }; // NOLINT template<> struct ConwayPolynomial<163, 6> { using ZPZ = aerobus::zpz<163>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<25>, ZPZV<156>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<163, 7> { using ZPZ = aerobus::zpz<163>; using type =
04053
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        template<> struct ConwayPolynomial<163, 8> { using ZPZ = aerobus::zpz<163>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<132>, ZPZV<83>, ZPZV<6>, ZPZV<2»; }; //
                                       template<> struct ConwayPolynomial<163, 9> { using ZPZ = aerobus::zpz<163>; using type =
04056
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<162>, ZPZV<127>, ZPZV<161»;
                         }; // NOLINT
                                       template<> struct ConwayPolynomial<163, 10> { using ZPZ = aerobus::zpz<163>; using type =
04057
                         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>
04058
                                        template<> struct ConwayPolynomial<163, 11> { using ZPZ = aerobus::zpz<163>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<11>, ZPZV<161»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type
04059
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<31>, ZPZV<38>, ZPZV<103>,
                         ZPZV<10>, ZPZV<69>, ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<163, 13> { using ZPZ = aerobus::zpz<163>; using type =
04060
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          template<> struct ConwayPolynomial<163,</pre>
                                                                                                                                                                                                             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<71>, ZPZV<161»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<163, 19> { using ZPZ = aerobus::zpz<163>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04062
                          ZPZV<0>, ZPZV<6</pre>
                         NOLINT
 04063
                                          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 =
04064
                         POLYV<ZPZV<1>, ZPZV<166>, ZPZV<5»; }; // NOLINT
                                         template<> struct ConwayPolynomial<167, 3> { using ZPZ = aerobus::zpz<167>; using type =
 04065
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162»; }; // NOLINT template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; }; // NOLINT
                                        template<> struct ConwayPolynomial<167, 5> { using ZPZ = aerobus::zpz<167>; using type =
 04067
                        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 =
04068
                         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
 04069
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<162»; }; // NOLINT template<> struct ConwayPolynomial<167, 8> { using ZPZ = aerobus::zpz<167>; using type =
 04070
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<149>, ZPZV<56>, ZPZV<113>, ZPZV<5»; }; //
                         NOLINT
04071
                                          template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<165>, ZPZV<162»;
                         }; // NOLINT
04072
                                       template<> struct ConwayPolynomial<167, 10> { using ZPZ = aerobus::zpz<167>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<68>, ZPZV<109>, ZPZV<143>,
                         ZPZV<148>, ZPZV<5»; };
                                                                                                                          // NOLINT
                                         template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type =
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<24>, ZPZV<162»; };
                                                                                                                           // NOLINT
04074
                                       template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<142>, ZPZV<10>, 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
04075
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162»; }; // NOLINT</pre>
04076
                                       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<0>, ZPZV<0>,
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162»; }; // NOLINT
   template<> struct ConwayPolynomial<167, 19> { using ZPZ = aerobus::zpz<167>; using type
04077
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<162»; }; //</pre>
                         NOLINT
 04078
                                        template<> struct ConwayPolynomial<173, 1> { using ZPZ = aerobus::zpz<173>; using type =
                       POLYV<ZPZV<1>, ZPZV<171»; }; // NOLINT
                                        template<> struct ConwayPolynomial<173, 2> { using ZPZ = aerobus::zpz<173>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<169>, ZPZV<2»; };
                                            template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
 04081
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<173, 5> { using ZPZ = aerobus::zpz<173>; using type =
04082
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171»; }; // NOLINT
                                            template<> struct ConwayPolynomial<173, 6> { using ZPZ = aerobus::zpz<173>; using type =
 04083
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<134>, ZPZV<107>, ZPZV<2»; }; // NOLINT
 04084
                                         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<0>, ZPZV<5>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 8> { using ZPZ = aerobus::zpz<173>; using type =
04085
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<125>, ZPZV<158>, ZPZV<27>, ZPZV<27>, ZPZV<2»; }; //
 04086
                                         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
04087
                                            template<> struct ConwayPolynomial<173, 10> { using ZPZ = aerobus::zpz<173>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<156>, ZPZV<164>, ZPZV<48>, ZPZV<106>,
                           ZPZV<58>, ZPZV<2»; };</pre>
                                                                                                                               // NOLINT
                                           template<> struct ConwayPolynomial<173, 11> { using ZPZ = aerobus::zpz<173>; using type =
                           \texttt{POLYV} < \texttt{ZPZV} < 0>, \ 
                           ZPZV<12>, ZPZV<171»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type =
04089
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<64>, ZPZV<46>, ZPZV<166>, ZPZV<0>,
                           ZPZV<159>, ZPZV<22>, ZPZV<2»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<173, 13> { using ZPZ = aerobus::zpz<173>; using type =
04090
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171»; };</pre>
                                                                                                                                                                                                           // NOLINT
                                          template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type =
04091
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<71*); };</pre>
                                                                                                                                                                                                                                                                                                                                                                             // NOLINT
                                            template<> struct ConwayPolynomial<173, 19> { using ZPZ = aerobus::zpz<173>; using type =
04092
                           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»; }; //</pre>
                           NOLINT
04093
                                            template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<177»; }; // NOLINT
                                             template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
                           POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2»; }; // NOLINT
04095
                                         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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<2»; }; // NOLINT
 04096
                                             template<> struct ConwayPolynomial<179, 5> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<177»; }; // NOLINT
 04098
                                         template<> struct ConwayPolynomial<179, 6> { using ZPZ = aerobus::zpz<179>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<91>, ZPZV<55>, ZPZV<109>, ZPZV<2»; }; // NOLINT
04099
                                           template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<177»; }; // NOLINT
                                            template<> struct ConwayPolynomial<179, 8> { using ZPZ = aerobus::zpz<179>; using type
04100
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<163>, ZPZV<144>, ZPZV<73>, ZPZV<23; };
                           NOLINT
04101
                                          template<> struct ConwayPolynomial<179, 9> { using ZPZ = aerobus::zpz<179>; using type =
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64>, ZPZV<64 , ZPZV<6
                           // NOLINT
                                            template<> struct ConwayPolynomial<179, 10> { using ZPZ = aerobus::zpz<179>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<115>, ZPZV<71>, ZPZV<150>, ZPZV<49>, ZPZV<87>,
                           ZPZV<2»; }; // NOLINT</pre>
04103
                                             template<> struct ConwayPolynomial<179, 11> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            template<> struct ConwayPolynomial<179, 12> { using ZPZ = aerobus::zpz<179>; using type
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<103>, ZPZV<43>, ZPZV<43>, ZPZV<76>, ZPZV<8>, ZPZV<177>, ZPZV<17, ZPZV<1, ZPZV<2»; }; // NOLINT
04105
                                         template<> struct ConwayPolynomial<179, 13> { using ZPZ = aerobus::zpz<179>; using type =
                           \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                           ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<177»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<179, 17> { using ZPZ = aerobus::zpz<179>; using type =
04106
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177»; };</pre>
                                         template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type =
04107
                           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<177»; }; //</pre>
                           NOLINT
                                            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
                                           template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
04110
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179»; }; // NOLINT template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
                         POLYV<ZPZV<1>, ZPZV<6>, ZPZV<65, ZPZV<105>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
 04112
                         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</pre>
 04113
```

```
template<> struct ConwayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<19»; }; // NOLINT template<> struct ConwayPolynomial<181, 8> { using ZPZ = aerobus::zpz<181>; using type =
04115
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<108>, ZPZV<22>, ZPZV<149>, ZPZV<2»; }; //
                       NOLINT
                                    template<> struct ConwayPolynomial<181, 9> { using ZPZ = aerobus::zpz<181>; using type =
04116
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<107>, ZPZV<168>, ZPZV<179»;
                       }; // NOLINT
04117
                                    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>
04118
                                    template<> struct ConwayPolynomial<181, 11> { using ZPZ = aerobus::zpz<181>; using type :
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   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</pre>
                                    template<> struct ConwayPolynomial<181, 13> { using ZPZ = aerobus::zpz<181>; using type =
04120
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<179»; };</pre>
                                                                                                                                                                                     // NOLINT
                                      template<> struct ConwayPolynomial<181, 17> { using ZPZ = aerobus::zpz<181>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0 , ZPZ
04122
04123
                                     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 =
04124
                      POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19»; }; // NOLINT
04125
                                      template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172»; }; // NOLINT
                                   template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
04126
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; }; // NOLINT template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
04127
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; }; // 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<19»; }; // NOLINT
                                    template<> struct ConwayPolynomial<191, 7> { using ZPZ = aerobus::zpz<191>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<14>; };
                                    template<> struct ConwayPolynomial<191, 8> { using ZPZ = aerobus::zpz<191>; using type =
04130
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<164>, ZPZV<139>, ZPZV<171>, ZPZV<19»; }; //
                                      template<> struct ConwayPolynomial<191, 9> { using ZPZ = aerobus::zpz<191>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<62>, ZPZV<124>, ZPZV<172»;
                       }; // NOLINT
                                      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
04133
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<6>, ZPZV<172»; }; // NOLINT</pre>
04134
                                     template<> struct ConwayPolynomial<191, 12> { using ZPZ = aerobus::zpz<191>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<79>, ZPZV<168>, ZPZV<25>, ZPZV<49>, ZPZV<90>,
                       ZPZV<7>, ZPZV<151>, ZPZV<19»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<191, 13> { using ZPZ = aerobus::zpz<191>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<172»; }; // NOLINT</pre>
04136
                                      template<> struct ConwayPolynomial<191,
                                                                                                                                                                                             17> { using ZPZ = aerobus::zpz<191>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<190>, ZPZV<190>, ZPZV<2>, ZPZV<172»; }; //</pre>
                       NOLINT
04138
                                      template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<188»; }; // NOLINT
                                     template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
04139
                      POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
04140
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
04141
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
04142
                      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 =
04143
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<8>, ZPZV<172>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<193, 7> { using ZPZ = aerobus::zpz<193>; using type =
04144
                      POLYV<ZPZV<1>, 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 =
04145
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<145>, ZPZV<34>, ZPZV<154>, ZPZV<5»; }; //
04146
                                    template<> struct ConwayPolynomial<193, 9> { using ZPZ = aerobus::zpz<193>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<168>, ZPZV<168>, ZPZV<27>, ZPZV<188»;
                       }; // NOLINT
04147
                                   template<> struct ConwayPolynomial<193, 10> { using ZPZ = aerobus::zpz<193>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<77>, ZPZV<89>,
                        ZPZV<5»; }; // NOLINT
04148
                                     template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type =
                        \texttt{POLYV} < \texttt{ZPZV} < \texttt{0} >, \ \texttt{ZPZV} < \texttt{
                        ZPZV<1>, ZPZV<188»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<193, 12> { using ZPZ = aerobus::zpz<193>; using type =
04149
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<5>, ZPZV<5>, ZPZV<135>, ZPZV<155>,
                        \text{ZPZV}<90>, \text{ZPZV}<46>, \text{ZPZV}<28>, \text{ZPZV}<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<193, 13> { using ZPZ = aerobus::zpz<193>; using type =
04150
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       template<> struct ConwayPolynomial<193, 17> { using ZPZ = aerobus::zpz<193>; using type
04151
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<188»; };</pre>
04152
                                     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<5>, ZPZV<5>, ZPZV<188»; }; //
                        NOLINT
                                        template<> struct ConwayPolynomial<197, 1> { using ZPZ = aerobus::zpz<197>; using type =
                       POLYV<ZPZV<1>, ZPZV<195»; }; // NOLINT
                                         template<> struct ConwayPolynomial<197, 2> { using ZPZ = aerobus::zpz<197>; using type =
                       POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2»; }; // NOLINT
                                       template<> struct ConwayPolynomial<197, 3> { using ZPZ = aerobus::zpz<197>; using type =
04155
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<195»; }; // NOLINT template<> struct ConwayPolynomial<197, 4> { using ZPZ = aerobus::zpz<197>; using type =
04156
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<124>, ZPZV<2»; }; // NOLINT
04157
                                       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
04158
                                        template<> struct ConwayPolynomial<197, 6> { using ZPZ = aerobus::zpz<197>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<124>, ZPZV<79>, ZPZV<173>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<197, 7> { using ZPZ = aerobus::zpz<197>; using type =
04159
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<195»; };
                                        template<> struct ConwayPolynomial<197, 8> { using ZPZ = aerobus::zpz<197>; using type
04160
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<29>, ZPZV<29>, ZPZV<2»; };
                        NOLINT
                                        template<> struct ConwayPolynomial<197, 9> { using ZPZ = aerobus::zpz<197>; using type =
04161
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<127>, ZPZV<8>, ZPZV<195»;
                                          template<> struct ConwayPolynomial<197, 10> { using ZPZ = aerobus::zpz<197>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<121>, ZPZV<137>, ZPZV<8>, ZPZV<73>, ZPZV<42>,
                        ZPZV<2»; }; // NOLINT</pre>
                       \label{eq:convayPolynomial} $$ \end{convayPolynomial} $$ 197, 11> { using ZPZ = aerobus::zpz<197>; using type = POLYV<ZPZV<1>, ZPZV<0>, 
04163
                        ZPZV<14>, ZPZV<195»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<197, 12> { using ZPZ = aerobus::zpz<197>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<168>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>,
                        ZPZV<90>, ZPZV<163>, ZPZV<2»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<197, 13> { using ZPZ = aerobus::zpz<197>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<195»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<197,
                                                                                                                                                                                                        17> { using ZPZ = aerobus::zpz<197>; using type
04166
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<195»; }; // NOLINT
    template<> struct ConwayPolynomial<197, 19> { using ZPZ = aerobus::zpz<197>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04167
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6</pre>, ZPZV<195»; }; //</pre>
                                       template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
                       POLYV<ZPZV<1>, ZPZV<196»; }; // NOLINT
04169
                                         template<> struct ConwayPolynomial<199, 2> { using ZPZ = aerobus::zpz<199>; using type =
                      POLYV<ZPZV<1>, ZPZV<193>, ZPZV<3»; }; // NOLINT

template<> struct ConwayPolynomial<199, 3> { using ZPZ = aerobus::zpz<199>; using type =
04170
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; }; // NOLINT template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; }; // NOLINT
                                       template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
04172
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
04173
                                       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
04174
                                         template<> struct ConwayPolynomial<199, 7> { using ZPZ = aerobus::zpz<199>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<39, ZPZV<196»; }; // NOLINT template<> struct ConwayPolynomial<199, 8> { using ZPZ = aerobus::zpz<199>; using type =
04175
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<160>, ZPZV<23>, ZPZV<159>, ZPZV<3»; }; //
                        NOLINT
                                       template<> struct ConwayPolynomial<199, 9> { using ZPZ = aerobus::zpz<199>; using type =
04176
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<177>, ZPZV<141>, ZPZV<196»;
                       }; // NOLINT
04177
                                        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
04178
                                         template<> struct ConwayPolynomial<199, 11> { using ZPZ = aerobus::zpz<199>; using type :
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<1>, ZPZV<196»; }; // NOLINT</pre>
04179
                                        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 =
04180
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<196»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<199, 17> { using ZPZ = aerobus::zpz<199>; using type =
04181
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<196»; }; // NOLINT
template<> struct ConwayPolynomial<199, 19> { using ZPZ = aerobus::zpz<199>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<196»; }; //</pre>
                                                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 =
04184
                             POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<209»; }; // NOLINT template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
 04186
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<211, 5> { using ZPZ = aerobus::zpz<211>; using type =
04187
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<209»; }; // NOLINT
                                                template<> struct ConwayPolynomial<211, 6> { using ZPZ = aerobus::zpz<211>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<81>, ZPZV<194>, ZPZV<133>, ZPZV<2»; }; // NOLINJ
04189
                                              template<> struct ConwayPolynomial<211, 7> { using ZPZ = aerobus::zpz<211>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2), ZPZV<2), ZPZV<2), ZPZV<20, ZPZV<20, ZPZV<20, ZPZV<20, ZPZV<21); using type =
04190
                              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<200>, ZPZV<87>, ZPZV<29>, ZPZV<29>; };
                                              template<> struct ConwayPolynomial<211, 9> { using ZPZ = aerobus::zpz<211>; using type =
04191
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<139>, ZPZV<139>, ZPZV<26>, ZPZV<209»;
                              }; // NOLINT
04192
                                                template<> struct ConwayPolynomial<211, 10> { using ZPZ = aerobus::zpz<211>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<30>, ZPZV<61>, ZPZV<148>, ZPZV<148>, ZPZV<87>, ZPZV<125>,
                              ZPZV<2»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<211, 11> { using ZPZ = aerobus::zpz<211>; using type
04193
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<7>, ZPZV<209»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type =
04194
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<80, ZPZV<145>, ZPZV<126>, ZPZV<184>, ZPZV<84>, ZPZV<27>, ZPZV<284>, ZPZV<284
                                                 template<> struct ConwayPolynomial<211,
                                                                                                                                                                                                                                                13> { using ZPZ = aerobus::zpz<211>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<209»; }; // NOLINT
    template<> struct ConwayPolynomial<211, 17> { using ZPZ = aerobus::zpz<211>; using type =
04196
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<20»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<211, 19> { using ZPZ = aerobus::zpz<211>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<209; }; //</pre>
                              NOLINT
                                                template<> struct ConwayPolynomial<223, 1> { using ZPZ = aerobus::zpz<223>; using type =
04198
                             POLYV<ZPZV<1>, ZPZV<220»; }; // NOLINT
                                                template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
                             POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; }; // NOLINT
 04200
                                              template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
04201
                            POLYV<ZPZV<1>, ZPZV<6>, ZPZV<65, ZPZV<163>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<223, 5> { using ZPZ = aerobus::zpz<223>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<223, 6> { using ZPZ = aerobus::zpz<223>; using type =
 04203
                              \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 2>, \ \texttt{ZPZV} < 68>, \ \texttt{ZPZV} < 24>, \ \texttt{ZPZV} < 196>, \ \texttt{ZPZV} < 3»; \ \}; \ \ // \ \ \texttt{NOLINT} 
                           template<> struct ConwayPolynomial<223, 7> { using ZPZ = aerobus::zpz<223>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 
04204
                                               template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<3»; }; //
04206
                                             template<> struct ConwayPolynomial<223, 9> { using ZPZ = aerobus::zpz<223>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<164>, ZPZV<64>, ZPZV<220»;
                              }; // NOLINT
                                                  template<> struct ConwayPolynomial<223, 10> { using ZPZ = aerobus::zpz<223>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<118>, ZPZV<177>, ZPZV<87>, ZPZV<89>, ZPZV<62>,
                               ZPZV<3»; }; // NOLINT</pre>
04208
                                             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 =
04209
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<64>, ZPZV<94>, ZPZV<11>, ZPZV<105>, ZPZV<64>,
                              ZPZV<151>, ZPZV<213>, ZPZV<3»; }; // NOLINT</pre>
04210
                                              template<> struct ConwayPolynomial<223, 13> { using ZPZ = aerobus::zpz<223>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                 template<> struct ConwayPolynomial<223, 17> { using ZPZ = aerobus::zpz<223>; using type =
04211
                               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<2>, ZPZV<2>, ZPZV<2>); // 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<7>, ZPZV<220»; }; //</pre>
                              NOT.TNT
```

```
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\ast; }; // NOLINT
 04215
                                          template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
                        POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<225»; }; // NOLINT template<> struct ConwayPolynomial<227, 4> { using ZPZ = aerobus::zpz<227>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2»; }; // NOLINT
 04217
                                         template<> struct ConwayPolynomial<227, 5> { using ZPZ = aerobus::zpz<227>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<25»; }; // NOLINT
                                         template<> struct ConwayPolynomial<227, 6> { using ZPZ = aerobus::zpz<227>; using type =
04218
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<24>, ZPZV<135>, ZPZV<2*; }; // NOLINT
04219
                                          template<> struct ConwayPolynomial<227,
                                                                                                                                                                                                               7> { using ZPZ = aerobus::zpz<227>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<225»; }; //
 04220
                                       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»; }; //
                          NOLTNT
                         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<2>, ZPZV<24>, ZPZV<24>, ZPZV<183>, ZPZV<24>, ZPZV<24>, ZPZV<25»;
04221
                         }; // NOLINT
template<> struct ConwayPolynomial<227, 10> { using ZPZ = aerobus::zpz<227>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<199>, ZPZV<12>, ZPZV<93>, ZPZV<77>,
                          ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type =
04223
                          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 =
04224
                         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 =
04225
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<225»; };</pre>
                                                                                                                                                                                                        // NOLINT
                                          template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; 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<8>, ZPZV<8>, ZPZV<225»; }; // NOLINT
template<> struct ConwayPolynomial<227, 19> { using ZPZ = aerobus::zpz<227>; using type =
04227
                          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<3+, ZPZV<3+, ZPZV<3+, ZPZV<3+</pre>; //
                                           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 =
04229
                         POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»: }: // NOLINT
                                         template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; }; // NOLINT template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
 04231
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<229, 5> { using ZPZ = aerobus::zpz<229>; using type =
04232
                         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 =
 04233
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<160>, ZPZV<186>, ZPZV<6»; };
                                          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<0>, ZPZV<2>, ZPZV<2
04235
                                         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<60»; }; //
                         NOLINT
                                          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<15>, ZPZV<117>, ZPZV<50>, ZPZV<223»;
                          }; // NOLINT
04237
                                          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<167>, ZPZV<98>, ZPZV<6»; }; // NOLINT
                                          template<> struct ConwayPolynomial<229, 11> { using ZPZ = aerobus::zpz<229>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                            // NOLINT
                          ZPZV<2>, ZPZV<223»; };</pre>
                        template<> struct ConwayPolynomial<229, 12> { using ZPZ = aerobus::zpz<229>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<131>, ZPZV<140>, ZPZV<25>, ZPZV<6>, ZPZV<172>,
04239
                          ZPZV<9>, ZPZV<145>, ZPZV<6»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                      // NOLINT
                          ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<223»; };</pre>
04241
                                       template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04242
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<228>, ZPZV<15>, ZPZV<223»; }; //</pre>
04243
                                          template<> struct ConwayPolynomial<233, 1> { using ZPZ = aerobus::zpz<233>; using type =
                         POLYV<ZPZV<1>. ZPZV<230»: 1: // NOLINT
                                          template<> struct ConwayPolynomial<233, 2> { using ZPZ = aerobus::zpz<233>; using type =
 04244
                         POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
                                          template<> struct ConwayPolynomial<233, 3> { using ZPZ = aerobus::zpz<233>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<230»; };
                                                                                                                                                                                                                                 // NOLINT
                                       template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
 04246
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<458>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<233, 5> { using ZPZ = aerobus::zpz<233>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<230»; };
                                    template<> struct ConwayPolynomial<233, 6> { using ZPZ = aerobus::zpz<233>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<122>, ZPZV<215>, ZPZV<32>, ZPZV<3»; }; // NOLINT
                                  template<> struct ConwayPolynomial<233, 7> { using ZPZ = aerobus::zpz<233>; using type =
04249
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 8> { using ZPZ = aerobus::zpz<233>; using type =
04250
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<202>, ZPZV<135>, ZPZV<181>, ZPZV<3»; }; //
                                  template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type =
04251
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<146>, ZPZV<230»;
                     }; // NOLINT
04252
                                   template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type :
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<28>, ZPZV<71>, ZPZV<102>, ZPZV<3>, ZPZV<48>,
                      ZPZV<3»; }; // NOLINT</pre>
04253
                                 template<> struct ConwayPolynomial<233, 11> { using ZPZ = aerobus::zpz<233>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<230»; }; // NOLINT
                                  template<> struct ConwayPolynomial<233, 12> { using ZPZ = aerobus::zpz<233>; using type :
04254
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<21>, ZPZV<114>, ZPZV<31>, ZPZV<19>,
                     ZPZV<216>, ZPZV<20>, ZPZV<3»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<233, 13> { using ZPZ = aerobus::zpz<233>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<230»; }; // NOLINT
template<> struct ConwayPolynomial<233, 17> { using ZPZ = aerobus::zpz<233>; using type =
04256
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<233,
                                                                                                                                                                              19> { using ZPZ = aerobus::zpz<233>; using type =
04257
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<25>, ZPZV<230»; }; //</pre>
                     NOLINT
                                  template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
04258
                     POLYV<ZPZV<1>, ZPZV<232»; }; // NOLINT
                                   template<> struct ConwayPolynomial<239, 2> { using ZPZ = aerobus::zpz<239>; using type =
                     POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7»; }; // NOLINT
 04260
                                   template<> struct ConwayPolynomial<239, 3> { using ZPZ = aerobus::zpz<239>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<232»; }; // NOLINT
template<> struct ConwayPolynomial<239, 4> { using ZPZ = aerobus::zpz<239>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7»; }; // NOLINT
04261
 04262
                                    template<> struct ConwayPolynomial<239, 5> { using ZPZ = aerobus::zpz<239>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; }; // NOLINT
04263
                                 template<> struct ConwayPolynomial<239, 6> { using ZPZ = aerobus::zpz<239>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<237>, ZPZV<60>, ZPZV<200>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<239, 7> { using ZPZ = aerobus::zpz<239>; using type = DVV<ZPZV<10.
04264
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<232»; };
                                   template<> struct ConwayPolynomial<239, 8> { using ZPZ = aerobus::zpz<239>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<202>, ZPZV<54>, ZPZV<7»; };
04266
                                 template<> struct ConwayPolynomial<239, 9> { using ZPZ = aerobus::zpz<239>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<28>, ZPZV<23x; };
                     // NOLINT
                                    template<> struct ConwayPolynomial<239, 10> { using ZPZ = aerobus::zpz<239>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>>, ZPZV<58>, ZPZV<266>, ZPZV<226>, ZPZV<127>,
                     ZPZV<108>, ZPZV<7»; }; // NOLINT</pre>
04268
                                   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>
                                    template<> struct ConwayPolynomial<239, 12> { using ZPZ = aerobus::zpz<239>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<235>, ZPZV<14>, ZPZV<113>, ZPZV<182>,
                     ZPZV<101>, ZPZV<81>, ZPZV<216>, ZPZV<7»; }; // NOLINT
  template<> struct ConwayPolynomial<239, 13> { using ZPZ = aerobus::zpz<239>; using type :
04270
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   template<> struct ConwayPolynomial<239, 17> { using ZPZ = aerobus::zpz<239>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<232»; };</pre>
                     template<> struct ConwayPolynomial<239, 19> { using ZPZ = aerobus::zpz<239>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2, ZPZV<0>, ZPZV<2, ZPZV<2, ZPZV<2, ZPZV<2, ZPZV<2, ZPZV<2, ZPZV<2, ZPZV<2, ZPZV<2, ZPZV<3, Z
04272
                     NOLINT
04273
                                    template<> struct ConwayPolynomial<241, 1> { using ZPZ = aerobus::zpz<241>; using type =
                     POLYV<ZPZV<1>, ZPZV<234»; }; // NOLINT
 04274
                                 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 =
 04275
                    POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<234»; }; // NOLINT template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<152>, ZPZV<7»; }; // NOLINT
                    template<> struct ConwayPolynomial<241, 5> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<234»; }; // NOLINT</pre>
 04277
04278
                                   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
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<234w; }; // NoLII template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
 04280
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<173>, ZPZV<212>, ZPZV<153>, ZPZV<19*; }; //
                     NOLINT
 04281
                                 template<> struct ConwayPolynomial<241, 9> { using ZPZ = aerobus::zpz<241>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<236>, ZPZV<125>, ZPZV<234»;
                       }; // NOLINT
04282
                                     template<> struct ConwayPolynomial<241, 10> { using ZPZ = aerobus::zpz<241>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<27>, ZPZV<145>, ZPZV<208>, ZPZV<55>,
                       ZPZV<7»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type =
04283
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                 // NOLINT
                       ZPZV<3>, ZPZV<234»; };</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
                                     template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type =
04285
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234»; }; // NOLINT</pre>
04286
                                   template<> struct ConwayPolynomial<241,
                                                                                                                                                                                             17> { using ZPZ = aerobus::zpz<241>; using type
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04287
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234»; }; //</pre>
04288
                                      template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
                      POLYV<ZPZV<1>, ZPZV<245»; }; // NOLINT
                                     template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
04289
                      POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; }; // NOLINT
                                       template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type =
04291
                      template<> struct ConwayPolynomial<251, 5> { using ZPZ = aerobus::zpz<251>; using type =
04292
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT
04293
                                       template<> struct ConwayPolynomial<251, 6> { using ZPZ = aerobus::zpz<251>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<247>, ZPZV<151>, ZPZV<179>, ZPZV<6»; }; // NOLINT
04294
                                   template<> struct ConwayPolynomial<251, 7> { using ZPZ = aerobus::zpz<251>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<45»; }; // NOLINT template<> struct ConwayPolynomial<251, 8> { using ZPZ = aerobus::zpz<251>; using type =
04295
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<142>, ZPZV<215>, ZPZV<173>, ZPZV<6»; }; //
                       NOLINT
04296
                                      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<48>, ZPZV<186>, ZPZV<106>, ZPZV<245»;
                       }; // NOLINT
04297
                      template<> struct ConwayPolynomial<251, 10> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<138>, ZPZV<110>, ZPZV<45>, ZPZV<34>,
                       ZPZV<149>, ZPZV<6»; };</pre>
                                                                                                                  // NOLINT
                                      template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<26>, ZPZV<245»; }; // NOLINT</pre>
                      template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<192>, ZPZV<53>, ZPZV<20>, ZPZV<20>, ZPZV<15>,
                       ZPZV<201>, ZPZV<232>, ZPZV<6»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<251, 13> { using ZPZ = aerobus::zpz<251>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<245»; }; // NOLINT</pre>
04301
                                      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<9>, ZPZV<245»; }; // NOLINT
template<> struct ConwayPolynomial<251, 19> { using ZPZ = aerobus::zpz<251>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8+, ZPZV<245»; }; //</pre>
                       NOLINT
04303
                                     template<> struct ConwayPolynomial<257, 1> { using ZPZ = aerobus::zpz<257>; using type =
                      POLYV<ZPZV<1>, ZPZV<254»; }; // NOLINT
04304
                                      template<> struct ConwayPolynomial<257, 2> { using ZPZ = aerobus::zpz<257>; using type =
                      POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3»; }; // NOLINT
04305
                                    template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65, 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 =
04306
04307
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<254»; }; // NOLINT
                                      template<> struct ConwayPolynomial<257, 6> { using ZPZ = aerobus::zpz<257>; using type =
04308
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<62>, ZPZV<18>, ZPZV<138>, ZPZV<3»; }; // NOLINT
                                      template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type
04309
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<254»; }; // NOLINT
                                      template<> struct ConwayPolynomial<257, 8> { using ZPZ = aerobus::zpz<257>; using type
04310
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<179>, ZPZV<140>, ZPZV<162>, ZPZV<3»; }; //
04311
                                     template<> struct ConwayPolynomial<257, 9> { using ZPZ = aerobus::zpz<257>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<50>, ZPZV<50>,
                       }; // NOLINT
04312
                                       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<97>, ZPZV<12>, ZPZV<225>, ZPZV<180>, ZPZV<20>,
                       ZPZV<3»; }; // NOLINT</pre>
04313
                                    template<> struct ConwayPolynomial<257, 11> { using ZPZ = aerobus::zpz<257>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04314
                                    template<> struct ConwayPolynomial<257, 12> { using ZPZ = aerobus::zpz<257>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<13>, ZPZV<225>, ZPZV<215>, ZPZV<173>,
                 ZPZV<249>, ZPZV<148>, ZPZV<20>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<257, 13> { using ZPZ = aerobus::zpz<257>; using type =
04315
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<254»; };</pre>
                             /<0>, ZPZV<0>, ZPZV<5>, ZPZV<254»; }; // NOLINT
template<> struct ConwayPolynomial<257, 17> { using ZPZ = aerobus::zpz<257>; using type =
04316
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<254»; };</pre>
                                                                                                                                                                                                                                                    // NOLINT
                            template<> struct ConwayPolynomial<257, 19> { using ZPZ = aerobus::zpz<257>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  NOLINT
                             template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
04318
                  POLYV<ZPZV<1>, ZPZV<258»; }; // NOLINT
 04319
                             template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
                 POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
04320
                 POLYY<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; }; // NOLINT template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; }; // NOLINT
                              template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<258»; }; // NOLINT
                            template<> struct ConwayPolynomial<263, 6> { using ZPZ = aerobus::zpz<263>; using type =
04323
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<22>, ZPZV<25>, ZPZV<25>, ZPZV<25>, ZPZV<263>; using ZPZ = aerobus::zpz<263>; using type
04324
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<258»; }; //
                            template<> struct ConwayPolynomial<263, 8> { using ZPZ = aerobus::zpz<263>; using type =
 04325
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<227>, ZPZV<170>, ZPZV<7>, ZPZV<5»; };
                  NOLINT
04326
                            template<> struct ConwayPolynomial<263, 9> { using ZPZ = aerobus::zpz<263>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<261>, ZPZV<261>, ZPZV<29>, ZPZV<258»;
                  }; // NOLINT
                             template<> struct ConwayPolynomial<263, 10> { using ZPZ = aerobus::zpz<263>; using type
04327
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<245>, ZPZV<241>, ZPZV<2198>, ZPZV<145>,
                  ZPZV<119>, ZPZV<5»; }; // NOLINT
  template<> struct ConwayPolynomial<263, 11> { using ZPZ = aerobus::zpz<263>; using type =
04328
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<258»; }; // NOLINT
                              template<> struct ConwayPolynomial<263, 12> { using ZPZ = aerobus::zpz<263>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<174>, ZPZV<162>, ZPZV<252>,
                 ZPZV<47>, ZPZV<45>, ZPZV<180>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
04330
                 POLYV<ZPZV<1>, ZPZV<267»; }; // NOLINT
04331
                             template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
                 POLYV<ZPZV<1>, ZPZV<268>, ZPZV<2»; }; // NOLINT
 04332
                             template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<267»; }; // NOLINT
template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<269, 5> { using ZPZ = aerobus::zpz<269>; using type =
04333
04334
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<267»; }; // NOLINT
                             template<> struct ConwayPolynomial<269, 6> { using ZPZ = aerobus::zpz<269>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<120>, ZPZV<101>, ZPZV<206>, ZPZV<2»; }; // NOLINT
04336
                            template<> struct ConwayPolynomial<269, 7> { using ZPZ = aerobus::zpz<269>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<267»; }; // NOLINT
                             template<> struct ConwayPolynomial<269, 8> { using ZPZ = aerobus::zpz<269>; using type =
 04337
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<220>, ZPZV<131>, ZPZV<232>, ZPZV<23x; }; //
                 template<> struct ConwayPolynomial<269, 9> { using ZPZ = aerobus::zpz<269>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<214>, ZPZV<267>, ZPZV<267>; ZPZV<267»;
                  }; // NOLINT
                 template<> struct ConwayPolynomial<269, 10> { using ZPZ = aerobus::zpz<269>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<264>, ZPZV<243>, ZPZV<186>, ZPZV<61>,
04339
                                                                                    // NOLINT
                  ZPZV<10>, ZPZV<2»; };</pre>
                            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>
                            template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type =
04341
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<165>, ZPZV<63>, ZPZV<215>,
                  ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type =
04342
                 POLYV<ZPZV<1>, ZPZV<265»; }; // NOLINT
                             template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
04343
                 POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; }; // NOLINT
                 template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT
04344
                            template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type =
 04345
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<271, 5> { using ZPZ = aerobus::zpz<271>; using type =
 04346
                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>; }; // NOLINT

template<> struct ConwayPolynomial<271, 6> { using ZPZ = aerobus::zpz<271>; using type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<00>, Z
 04347
                             template<> struct ConwayPolynomial<271, 7> { using ZPZ = aerobus::zpz<271>; using type
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<22>, ZPZV<265»; };
04349
                           template<> struct ConwayPolynomial<271, 8> { using ZPZ = aerobus::zpz<271>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<119>, ZPZV<114>, ZPZV<69>, ZPZV<6»; }; //
                  NOLTNT
```

```
template<> struct ConwayPolynomial<271, 9> { using ZPZ = aerobus::zpz<271>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<186>, ZPZV<186>, ZPZV<265»;
               }; // NOLINT
               template<> struct ConwayPolynomial<271, 10> { using ZPZ = aerobus::zpz<271>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<10>, ZPZV<256>, ZPZV<74>,
04351
               ZPZV<126>, ZPZV<6»; };</pre>
                                                                              // NOLINT
                          template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                ZPZV<10>, ZPZV<265»; }; // NOLINT</pre>
               template<> struct ConwayPolynomial<271, 12> { using ZPZ = aerobus::zpz<271>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162>, ZPZV<210>, ZPZV<116>, ZPZV<205>,
ZPZV<237>, ZPZV<256>, ZPZV<130>, ZPZV<6»; }; // NOLINT</pre>
04353
                          template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
04354
               POLYV<ZPZV<1>, ZPZV<272»; }; // NOLINT
04355
                         template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
04356
               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZV<3>, ZPZV<3 , ZPZV<3
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<5»; }; // NOLINT
                           template<> struct ConwayPolynomial<277, 5> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<272»; }; // NOLINT
                         template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type =
04359
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<9>, ZPZV<18>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type
04360
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<272»; }; // NOLINT
                         template<> struct ConwayPolynomial<277, 8> { using ZPZ = aerobus::zpz<277>; using type
04361
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<159>, ZPZV<176>, ZPZV<15»; }; //
               NOLINT
04362
                         template<> struct ConwayPolynomial<277, 9> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177>, ZPZV<110>, ZPZV<272»;
               }; // NOLINT
                          template<> struct ConwayPolynomial<277, 10> { using ZPZ = aerobus::zpz<277>; using type =
04363
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<206>, ZPZV<253>, ZPZV<237>, ZPZV<241>,
               ZPZV<260>, ZPZV<5»; }; // NOLINT
  template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type =
04364
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           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<180>, ZPZV<210>, ZPZV<202>, ZPZV<5»; ); // NOLINT
                          template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
04366
               POLYV<ZPZV<1>, ZPZV<278»; }; // NOLINT
04367
                         template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; }; // NOLINT
04368
                          template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218,; }; // NOLINT
template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<281, 5> { using ZPZ = aerobus::zpz<281>; using type =
04369
04370
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<278»; }; // NOLINT
                          template<> struct ConwayPolynomial<281, 6> { using ZPZ = aerobus::zpz<281>; using type =
                \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 151>, \ \texttt{ZPZV} < 13>, \ \texttt{ZPZV} < 27>, \ \texttt{ZPZV} < 3>; \ \}; \ // \ \texttt{NOLINT} 
04372
                         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<19>, ZPZV<278»; }; // NOLINT
                          template<> struct ConwayPolynomial<281, 8> { using ZPZ = aerobus::zpz<281>; using type =
04373
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195>, ZPZV<279>, ZPZV<140>, ZPZV<3»; }; //
               template<> struct ConwayPolynomial<281, 9> { using ZPZ = aerobus::zpz<281>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<148>, ZPZV<70>, ZPZV<278»;
               }; // NOLINT
               template<> struct ConwayPolynomial<281, 10> { using ZPZ = aerobus::zpz<281>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<145>, ZPZV<13>, ZPZV<138>,
04375
                                                                           // NOLINT
               ZPZV<191>, ZPZV<3»; };</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
04377
                         template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<PPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<202>, ZPZV<68>, ZPZV<103>, ZPZV<116>,
               ZPZV<58>, ZPZV<28>, ZPZV<191>, ZPZV<3»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
04378
               POLYV<ZPZV<1>, ZPZV<280»; }; // NOLINT
                          template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
04379
               POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3»; }; // NOLINT
                          template<> struct ConwayPolynomial<283, 3> { using ZPZ = aerobus::zpz<283>; using type =
04380
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
                         template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type =
04381
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
04382
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
              template<> struct ConwayPolynomial<283, 6> { using ZPZ = aerobus::zpz<283>; using type = PoLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<68>, ZPZV<73>, ZPZV<3»; }; // NOLINT
04383
                          template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<280»; };
04385
                        template<> struct ConwayPolynomial<283, 8> { using ZPZ = aerobus::zpz<283>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<179>, ZPZV<32>, ZPZV<232>, ZPZV<3»; }; //
               NOLTNT
```

```
template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                }; // NOLINT
                template<> struct ConwayPolynomial<283, 10> { using ZPZ = aerobus::zpz<283>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<271>, ZPZV<185>, ZPZV<68>, ZPZV<100>,
04387
                ZPZV<219>, ZPZV<3»; };</pre>
                                                                                 // NOLINT
                           template<> struct ConwayPolynomial<283, 11> { using ZPZ = aerobus::zpz<283>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                 ZPZV<4>, ZPZV<280»; }; // NOLINT</pre>
                template<> struct ConwayPolynomial<283, 12> { using ZPZ = aerobus::zpz<283>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<8>, ZPZV<8>, ZPZV<96>, ZPZV<29>, ZPZV<49>,
ZPZV<14>, ZPZV<56>, ZPZV<3»; }; // NOLINT</pre>
04389
                           template<> struct ConwayPolynomial<293, 1> { using ZPZ = aerobus::zpz<293>; using type =
04390
                POLYV<ZPZV<1>, ZPZV<291»; }; // NOLINT
04391
                           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 =
04392
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
                           template<> struct ConwayPolynomial2293, 4> { using ZPZ = aerobus::zpz<293>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2»; }; // NOLINT
                            template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
                          template<> struct ConwayPolynomial<293, 6> { using ZPZ = aerobus::zpz<293>; using type =
04395
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<10>, ZPZV<260>, ZPZV<28>; // NOLINT template<> struct ConwayPolynomial<293, 7> { using ZPZ = aerobus::zpz<293>; using type
04396
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<291»; }; //
                          template<> struct ConwayPolynomial<293, 8> { using ZPZ = aerobus::zpz<293>; using type =
04397
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<2»; }; //
                NOLINT
04398
                          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
                          template<> struct ConwayPolynomial<293, 10> { using ZPZ = aerobus::zpz<293>; using type
04399
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<186>, ZPZV<28>, ZPZV<46>, ZPZV<184>, ZPZV<24>,
                 ZPZV<2»; }; // NOLINT</pre>
04400
                           template<> struct ConwayPolynomial<293, 11> { using ZPZ = aerobus::zpz<293>; using type =
                POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                            template<> struct ConwayPolynomial<293, 12> { using ZPZ = aerobus::zpz<293>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2), ZPZV<2159>, ZPZV<210>, ZPZV<125>, ZPZV<212>, ZPZV<167>, ZPZV<144>, ZPZV<157>, ZPZV<22*; }; // NOLINT template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
04402
                POLYV<ZPZV<1>, ZPZV<302»; }; // NOLINT
                          template<> struct ConwayPolynomial<307, 2> { using ZPZ = aerobus::zpz<307>; using type =
04403
                POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5»; }; // NOLINT
04404
                           template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<30>, ; // NOLINT
template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5>; // NOLINT
template<> struct ConwayPolynomial<307, 5> { using ZPZ = aerobus::zpz<307>; using type =
04405
04406
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<302»; }; // NOLINT
                           template<> struct ConwayPolynomial<307, 6> { using ZPZ = aerobus::zpz<307>; using type =
04407
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<213>, ZPZV<172>, ZPZV<61>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<307, 7> { using ZPZ = aerobus::zpz<307>; using type =
POLYV<ZPZV<1>, 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 =
04408
04409
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<283>, ZPZV<232>, ZPZV<131>, ZPZV<5»; }; //
                template<> struct ConwayPolynomial<307, 9> { using ZPZ = aerobus::zpz<307>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<165>, ZPZV<165>, ZPZV<302»;
04410
                }; // NOLINT
                           template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
04411
                POLYV<ZPZV<1>, ZPZV<294»; }; // NOLINT
                           template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
                POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; }; // NOLINT
                          template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
04413
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; }; // NOLINT
template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; }; // NOLINT
04414
                           template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
04415
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294»; }; // NOLINT
04416
                         template<> struct ConwayPolynomial<311, 6> { using ZPZ = aerobus::zpz<311>; using type =
               04417
                POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3, ZPZV<0>, ZPZV<3, ZPZV<3,
                                                                                                                                                                                                                                                    // NOLINT
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<162>, ZPZV<118>, ZPZV<2>, ZPZV<217»; };
04419
                          template<> struct ConwayPolynomial<311, 9> { using ZPZ = aerobus::zpz<311>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<287>, ZPZV<294»;
                 }; // NOLINT
                           template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
                POLYV<ZPZV<1>, ZPZV<303»; }; // NOLINT
04421
                          template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
               POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
04422
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; }; // NOLINT
```

```
04423
               template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<8>, ZPZV<239>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
04424
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<303»; }; // NOLINT
        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 =
04425
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<303»; };
04427
              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 =
04428
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<3
         }; // NOLINT
04429
               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 =
04430
         POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<315»; }; // NOLINT template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<317, 5> { using ZPZ = aerobus::zpz<317>; using type =
04433
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<315»; // NOLINT
               template<> struct ConwayPolynomial<317, 6> { using ZPZ = aerobus::zpz<317>; using type =
04434
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<195>, ZPZV<156>, ZPZV<4>, ZPZV<4»; }; // NOLINT
              template<> struct ConwayPolynomial<317, 7> { using ZPZ = aerobus::zpz<317>; using type =
04435
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<315»; }; // NOLINT template<> struct ConwayPolynomial<317, 8> { using ZPZ = aerobus::zpz<317>; using type =
04436
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<207>, ZPZV<85>, ZPZV<31>, ZPZV<2»; };
         NOLINT
04437
               template<> struct ConwayPolynomial<317, 9> { using ZPZ = aerobus::zpz<317>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<284>, ZPZV<296>, ZPZV<315»;
         }; // NOLINT
04438
               template<> struct ConwayPolynomial<331, 1> { using ZPZ = aerobus::zpz<331>; using type =
         POLYV<ZPZV<1>, ZPZV<328»; }; // NOLINT
               template<> struct ConwayPolynomial<331, 2> { using ZPZ = aerobus::zpz<331>; using type =
04439
         POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<328»; }; // NOLINT template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
04441
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<290>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<331, 5> { using ZPZ = aerobus::zpz<331>; using type =
04442
         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»; };
04444
              template<> struct ConwayPolynomial<331, 7> { using ZPZ = aerobus::zpz<331>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<328»; }; // NOLINT
04445
              template<> struct ConwayPolynomial<331, 8> { using ZPZ = aerobus::zpz<331>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<24>, ZPZV<308>, ZPZV<78>, ZPZV<3»; };
               template<> struct ConwayPolynomial<331, 9> { using ZPZ = aerobus::zpz<331>; using type =
04446
         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
04447
               template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
         POLYV<ZPZV<1>, ZPZV<327»; }; // NOLINT
               template<> struct ConwayPolynomial<337, 2> { using ZPZ = aerobus::zpz<337>; using type =
         POLYV<ZPZV<1>, ZPZV<332>, ZPZV<10»; }; // NOLINT
               template<> struct ConwayPolynomial<337, 3> { using ZPZ = aerobus::zpz<337>; using type =
04449
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
04450
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<225>, ZPZV<210»; }; // NOLINT template<> struct ConwayPolynomial<337, 5> { using ZPZ = aerobus::zpz<337>; using type =
04451
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT
04452
              template<> struct ConwayPolynomial<337, 6> { using ZPZ = aerobus::zpz<337>; using type =
         POLYV<ZPZV<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
04453
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<327»; }; // NOLINT
              template<> struct ConwayPolynomial<337, 8> { using ZPZ = aerobus::zpz<337>; using type =
04454
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<331>, ZPZV<246>, ZPZV<251>, ZPZV<10»; };
04455
              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
04456
               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 =
04457
         POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
04458
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<345»; }; // NOLINT
template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; }; // NOLINT
04459
               template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<345»; }; // NOLINT
04461
              template<> struct ConwayPolynomial<347, 6> { using ZPZ = aerobus::zpz<347>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<343>, ZPZV<26>, ZPZV<56>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
04462
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<345»; };
               template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<187>, ZPZV<213>, ZPZV<117>, ZPZV<2»; }; //
         NOLINT
04464
               template<> struct ConwayPolynomial<347, 9> { using ZPZ = aerobus::zpz<347>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<25>, ZPZV<25>, ZPZV<345»;
04465
               template<> struct ConwayPolynomial<349, 1> { using ZPZ = aerobus::zpz<349>; using type =
        POLYV<ZPZV<1>, ZPZV<347»; }; // NOLINT
              template<> struct ConwayPolynomial<349, 2> { using ZPZ = aerobus::zpz<349>; using type =
04466
        POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
04467
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; }; // NOLINT template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2»; }; // NOLINT
        template<> struct ConwayPolynomial<349, 5> { using ZPZ = aerobus::zpz<349>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<347»; }; // NOLINT</pre>
04469
               template<> struct ConwayPolynomial<349, 6> { using ZPZ = aerobus::zpz<349>; using type =
04470
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<135>, ZPZV<177>, ZPZV<316>, ZPZV<2»; }; // NOLINT
04471
               template<> struct ConwayPolynomial<349,
                                                                           7> { using ZPZ = aerobus::zpz<349>; using type
         POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<10>, 2PZV<347»; }; //
04472
               template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<328>, ZPZV<268>, ZPZV<268; }; //
         NOLINT
04473
               template<> struct ConwayPolynomial<349, 9> { using ZPZ = aerobus::zpz<349>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<290>, ZPZV<130>, ZPZV<347»;
         }; // NOLINT
04474
              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 =
04475
        POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3»; }; // NOLINT
04476
               template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZV<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type =
04477
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
04478
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<1>, ZPZY<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 6> { using ZPZ = aerobus::zpz<353>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<215>, ZPZV<226>, ZPZV<295>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<353, 7> { using ZPZ = aerobus::zpz<353>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<350»; };
              template<> struct ConwayPolynomial<353, 8> { using ZPZ = aerobus::zpz<353>; using type =
04481
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<182>, ZPZV<26>, ZPZV<37>, ZPZV<3»; };
               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<0>, ZPZV<2>, ZPZV<319>, ZPZV<49>, ZPZV<350»;
         }; // NOLINT
04483
               template<> struct ConwayPolynomial<359, 1> { using ZPZ = aerobus::zpz<359>; using type =
        POLYV<ZPZV<1>, ZPZV<352»; }; // NOLINT
               template<> struct ConwayPolynomial<359, 2> { using ZPZ = aerobus::zpz<359>; using type =
04484
        POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7»; }; // NOLINT
               template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<352»; }; // NOLINT template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
04486
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<229>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<359, 5> { using ZPZ = aerobus::zpz<359>; using type =
04487
         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 =
04488
        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 =
04489
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>; }; // NOLINT template<> struct ConwayPolynomial<359, 8> { using ZPZ = aerobus::zpz<359>; using type =
04490
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<301>, ZPZV<143>, ZPZV<271>, ZPZV<7»; }; //
04491
              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<0>, ZPZV<356>, ZPZV<165>, ZPZV<352»;
         }; // NOLINT
               \texttt{template<> struct ConwayPolynomial<367, 1> \{ using ZPZ = aerobus:: zpz<367>; using type = 200 aerobus:: zpz<367>; usin
04492
        POLYV<ZPZV<1>, ZPZV<361»; }; // NOLINT
04493
               template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
         POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6»; }; // NOLINT
04494
              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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6»; }; // NOLINT

template<> struct ConwayPolynomial<367, 5> { using ZPZ = aerobus::zpz<367>; using type =
04495
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<361»; }; // NOLINT
04497
               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<66; }; // NOLINT template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type :
04498
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361»; }; // NOLINT
               template<> struct ConwayPolynomial<367, 8> { using ZPZ = aerobus::zpz<367>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<335>, ZPZV<282>, ZPZV<50>, ZPZV<60; };
         NOLINT
04500
              template<> struct ConwayPolynomial<367, 9> { using ZPZ = aerobus::zpz<367>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<215>, ZPZV<213>, ZPZV<268>, ZPZV<361»;
         }; // NOLINT
```

```
04501
                 template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
          POLYV<ZPZV<1>, ZPZV<371»; }; // NOLINT
04502
                template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
          POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2»; }; // NOLINT
04503
                 template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
         POLYV<ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<571; }; // 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
                template<> struct ConwayPolynomial<373, 5> { using ZPZ = aerobus::zpz<373>; using type =
04505
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<371»; }; // NOLINT
04506
                template<> struct ConwayPolynomial<373, 6> { using ZPZ = aerobus::zpz<373>; using type =
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<83>, ZPZV<108>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type
04507
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7
04508
                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»; };
          NOT.TNT
          template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<370>, ZPZ
04509
04510
                 template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
          POLYV<ZPZV<1>, ZPZV<377»; }; // NOLINT
                template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
04511
          POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<379, 3> { using ZPZ = aerobus::zpz<379>; using type =
04512
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<577%; }; // NOLINT template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
04513
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<379, 5> { using ZPZ = aerobus::zpz<379>; using type =
04514
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<377»; }; // NOLINT template<> struct ConwayPolynomial<379, 6> { using ZPZ = aerobus::zpz<379>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<374>, ZPZV<246>, ZPZV<246>, ZPZV<246>; // NOLINT
04515
                 template<> struct ConwayPolynomial<379, 7> { using ZPZ = aerobus::zpz<379>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<377»; };
          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<173>, ZPZV<2»; }; //</pre>
          NOLINT
                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
04519
                 template<> struct ConwayPolynomial<383, 1> { using ZPZ = aerobus::zpz<383>; using type =
          POLYV<ZPZV<1>, ZPZV<378»; }; // NOLINT
                 template<> struct ConwayPolynomial<383, 2> { using ZPZ = aerobus::zpz<383>; using type =
04520
          POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; }; // NOLINT
                 template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<378»; };
                                                                                            // NOLINT
                template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
04522
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<383, 5> { using ZPZ = aerobus::zpz<383>; using type =
04523
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
04524
                 template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<8>, ZPZV<158>, ZPZV<158>, ZPZV<5»; }; // NOLINT
04525
                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<6>, ZPZV<6>, ZPZV<6>, ZPZV<378»; }; // NOLINT template<> struct ConwayPolynomial<383, 8> { using ZPZ = aerobus::zpz<383>; using type =
04526
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<281>, ZPZV<332>, ZPZV<296>, ZPZV<5»; }; //
04527
                template<> struct ConwayPolynomial<383, 9> { using ZPZ = aerobus::zpz<383>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137>, ZPZV<16>, ZPZV<378»;
          }; // NOLINT
04528
                template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
          POLYV<ZPZV<1>, ZPZV<387»; }; // NOLINT
04529
                 template<> struct ConwayPolynomial<389, 2> { using ZPZ = aerobus::zpz<389>; using type =
          POLYV<ZPZV<1>, ZPZV<379>, ZPZV<2»; }; // NOLINT
                template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
04530
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387»; }; // NOLINT
template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<266>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
04531
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<387»; }; // NOLINT
                 template<> struct ConwayPolynomial<389, 6> { using ZPZ = aerobus::zpz<389>; using type =
04533
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<339>, ZPZV<255>, ZPZV<2»; }; // NOLINT
04534
                 template<> struct ConwayPolynomial<389, 7> { using ZPZ = aerobus::zpz<389>; using type
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<24>, ZPZV<387»; }; // NOLINT template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::zpz<389>; using type =
04535
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<351>, ZPZV<19>, ZPZV<290>, ZPZV<2°; };
04536
                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
04537
                 template<> struct ConwayPolynomial<397, 1> { using ZPZ = aerobus::zpz<397>; using type =
         POLYV<ZPZV<1>, ZPZV<392»; };
                                                            // NOLINT
                 template<> struct ConwayPolynomial<397, 2> { using ZPZ = aerobus::zpz<397>; using type =
         POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5»; }; // NOLINT
                template<> struct ConwayPolynomial<397, 3> { using ZPZ = aerobus::zpz<397>; using type =
04539
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<392»; }; // NOLINT template<> struct ConwayPolynomial<397, 4> { using ZPZ = aerobus::zpz<397>; using type =
04540
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5»; };
                    template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392»; }; // NOLINT
                   template<> struct ConwayPolynomial<397, 6> { using ZPZ = aerobus::zpz<397>; using type =
04542
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<274>, ZPZV<287>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 7> { using ZPZ = aerobus::zpz<397>; using type
04543
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<392»; }; // NOLINT
04544
                   template<> struct ConwayPolynomial<397, 8> { using ZPZ = aerobus::zpz<397>;
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<375>, ZPZV<255>, ZPZV<203>, ZPZV<5»; }; //
04545
                  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<6>, ZPZV<166>, ZPZV<166>, ZPZV<252>, ZPZV<392»;
           }; // NOLINT
   template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
           POLYV<ZPZV<1>, ZPZV<398»; }; // NOLINT
04547
                   template<> struct ConwayPolynomial<401, 2> { using ZPZ = aerobus::zpz<401>; using type =
           POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
04548
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<398»; }; // NOLINT
                   template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<3»; }; // NOLINT
04550
                   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 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
04551
                    template<> struct ConwayPolynomial<401, 7> { using ZPZ = aerobus::zpz<401>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<398»; };
                  template<> struct ConwayPolynomial<401, 8> { using ZPZ = aerobus::zpz<401>; using type =
04553
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<380>, ZPZV<113>, ZPZV<164>, ZPZV<3»; }; //
           NOLINT
           template<> struct ConwayPolynomial<401, 9> { using ZPZ = aerobus::zpz<401>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<198>, ZPZV<398»;
04554
           }; // NOLINT
                   template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
04555
           POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
                   template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
04556
           POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT
                    template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388»; }; // NOLINT
                   template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
04558
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<407>, ZPZV<21»; }; // NOLINT template<> struct ConwayPolynomial<409, 5> { using ZPZ = aerobus::zpz<409>; using type =
04559
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<388»; }; // NOLINT
04560
                   template<> struct ConwayPolynomial<409, 6> { using ZPZ = aerobus::zpz<409>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<372>, ZPZV<53>, ZPZV<364>, ZPZV<21»; }; // NOLINT
04561
                   template<> struct ConwayPolynomial<409, 7> { using ZPZ = aerobus::zpz<409>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<388»; }; // NOLINT template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
04562
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<256>, ZPZV<69>, ZPZV<3965, ZPZV<
           NOLINT
04563
                   template<> struct ConwayPolynomial<409, 9> { using ZPZ = aerobus::zpz<409>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<818>, ZPZV<318>, ZPZV<211>, ZPZV<388»;
           }; // NOLINT
04564
                   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 =
04565
           POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
          template<> struct ConwayPolynomial

POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<417»; }; // NOLINT
template<> struct ConwayPolynomial

### A Susing ZPZ = aerobus::zpz<419>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<373>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial

### A Susing ZPZ = aerobus::zpz<419>; 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<0</pre>

### A Susing ZPZ = aerobus::zpz<419>; using type = POLYV<ZPZV<0</pre>

### A Susing ZPZ = aerobus::zpz<419>; using type = POLYV
04567
04568
           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 =
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<411>, ZPZV<33>, ZPZV<257>, ZPZV<2»; }; // NOLINT
04570
                  template<> struct ConwayPolynomial<419, 7> { using ZPZ = aerobus::zpz<419>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 8> { using ZPZ = aerobus::zpz<419>; using type =
04571
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<34>, ZPZV<388>, ZPZV<151>, ZPZV<2»; }; //
           NOLINT
                   template<> struct ConwayPolynomial<419, 9> { using ZPZ = aerobus::zpz<419>; using type =
04572
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
           }; // NOLINT
04573
                    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 =
           POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT
04575
                   template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419»; }; // NOLINT template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
04576
           POLYVCZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<25; }; // NOLINT template<> struct ConwayPolynomial<421, 5> { using ZPZ = aerobus::zpz<421>; using type =
04577
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<419»; }; // NOLINT
04578
                  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<21>, ZPZV<419»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<421, 8> { using ZPZ = aerobus::zpz<421>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<389>, ZPZV<32>, ZPZV<77>, ZPZV<2»; }; //
         NOLINT
        template<> struct ConwayPolynomial<421, 9> { using ZPZ = aerobus::zpz<421>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<394>, ZPZV<145>, ZPZV<419»;</pre>
04581
         }; // NOLINT
04582
               template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
         POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
              template<> struct ConwayPolynomial<431, 2> { using ZPZ = aerobus::zpz<431>; using type =
04583
        POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<431, 3> { using ZPZ = aerobus::zpz<431>; using type =
04584
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424»; }; // NOLINT template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
04585
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7»; }; // NOLINT
04586
              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 =
04587
        POLYYCZPZV<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
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<424»; };
               template<> struct ConwayPolynomial<431, 8> { using ZPZ = aerobus::zpz<431>; using type =
         POLYV<2PZV<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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<71>, ZPZV<329>, ZPZV<424*;</pre>
04590
         }; // NOLINT
              template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
04591
        POLYV<ZPZV<1>, ZPZV<428»; }; // NOLINT
04592
               template<> struct ConwayPolynomial<433, 2> { using ZPZ = aerobus::zpz<433>; using type =
        POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<433, 3> { using ZPZ = aerobus::zpz<433>; using type =
04593
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<428»; }; // NOLINT template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
04594
        04595
               template<> struct ConwayPolynomial<433, 5> { using ZPZ = aerobus::zpz<433>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<428»; }; // NOLINT
        template<> struct ConwayPolynomial<433, 6> { using ZPZ = aerobus::zpz<433>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<244>, ZPZV<360>, ZPZV<360>, ZPZV<5»; }; // NOLINT
04596
04597
               template<> struct ConwayPolynomial<433, 7> { using ZPZ = aerobus::zpz<433>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
04598
              template<> struct ConwayPolynomial<433, 8> { using ZPZ = aerobus::zpz<433>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347>, ZPZV<32>, ZPZV<39>, ZPZV<5»; };
         NOLINT
04599
              template<> struct ConwayPolynomial<433, 9> { using ZPZ = aerobus::zpz<433>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<27>, ZPZV<232>, ZPZV<45>, ZPZV<428»;
         }; // NOLINT
04600
              template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
               template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
04601
         POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; }; // NOLINT
               template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
04602
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424»; }; // NOLINT
04603
               template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; }; // NOLINT template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
04604
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT
               template<> struct ConwayPolynomial<439, 6> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<324>, ZPZV<190>, ZPZV<15»; }; // NOLINT
               template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLIN template<> struct ConwayPolynomial<439, 8> { using ZPZ = aerobus::zpz<439>; using type =
04607
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<359>, ZPZV<296>, ZPZV<266>, 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<16>, ZPZV<342>, ZPZV<254>, ZPZV<424»;
         }; // NOLINT
04609
               template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
        POLYV<ZPZV<1>, ZPZV<441»; }; // NOLINT
               template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
04610
        POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
04611
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
04612
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<443>; using type =
04613
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT
               template<> struct ConwayPolynomial<443, 6> { using ZPZ = aerobus::zpz<443>; using type =
04614
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<298>, ZPZV<218>, ZPZV<41>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<443, 7> { using ZPZ = aerobus::zpz<443>; using type =
04615
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6441»; }; // NOLINT
              template<> struct ConwayPolynomial<443, 8> { using ZPZ = aerobus::zpz<443>; using type
04616
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<217>, ZPZV<290>, ZPZV<2»; }; //
04617
              template<> struct ConwayPolynomial<4443, 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
04618
              template<> struct ConwayPolynomial<449, 1> { using ZPZ = aerobus::zpz<449>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<446»; };
              template<> struct ConwayPolynomial<449, 2> { using ZPZ = aerobus::zpz<449>; using type =
        POLYV<ZPZV<1>, ZPZV<444>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<449, 3> { using ZPZ = aerobus::zpz<449>; using type =
04620
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446»; }; // NOLINT template<> struct ConwayPolynomial<449, 4> { using ZPZ = aerobus::zpz<449>; using type =
04621
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3»; }; // NOLINT
04622
              template<> struct ConwayPolynomial<449, 5> { using ZPZ = aerobus::zpz<449>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<446»; }; // NOLINT
        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 =
04623
04624
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<28>, ZPZV<446»; };
              template<> struct ConwayPolynomial<449, 8> { using ZPZ = aerobus::zpz<449>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361>, ZPZV<348>, ZPZV<124>, ZPZV<3»; }; //
             template<> struct ConwayPolynomial<449, 9> { using ZPZ = aerobus::zpz<449>; using type =
04626
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<446»; };
             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 =
04628
        POLYV<ZPZV<1>, ZPZV<454>, ZPZV<13»; // NOLINT
             template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
04629
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444*; }; // NOLINT
              template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13»; }; // NOLINT
04631
              template<> struct ConwayPolynomial<457, 5> { using ZPZ = aerobus::zpz<457>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>; }; // NOLINT
        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 =
04632
04633
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<444»; };
04634
             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<8444»;
04635
        }; // NOLINT
              template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
04636
        POLYV<ZPZV<1>, ZPZV<459»; }; // NOLINT
              template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
04637
        POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2»: }: // NOLINT
              template<> struct ConwayPolynomial<461, 3> { using ZPZ = aerobus::zpz<461>; using type =
04638
        POLYV<ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<459»; }; // NOLINT template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
04639
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
04640
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<459»; }; // NOLINT
        template<> struct ConwayPolynomial</br>
template<> struct ConwayPolynomial</br>
template<> struct ConwayPolynomial</br>
template<> struct ConwayPolynomial</br>
template<> struct Service</br>
template<> struct Service</br>
template<> struct Service</br>
template<> struct Service
template
template</td
04641
              template<> struct ConwayPolynomial<461, 7> { using ZPZ = aerobus::zpz<461>; using type =
04642
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<459»; }; // NOLINT
04643
             template<> struct ConwayPolynomial<461, 8> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, 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 =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<210>, ZPZV<276>, ZPZV<459»;
        }; // NOLINT
04645
              template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
        POLYV<ZPZV<1>, ZPZV<460»; }; // NOLINT
             template<> struct ConwayPolynomial<463, 2> { using ZPZ = aerobus::zpz<463>; using type =
04646
        POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<460»; }; // NOLINT
template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
04648
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
04649
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<460»; }; // NOLINT
04650
              template<> struct ConwayPolynomial<463, 6> { using ZPZ = aerobus::zpz<463>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<462>, ZPZV<51>, ZPZV<110>, ZPZV<3»; }; // NOLINT
04651
             template<> struct ConwayPolynomial<463, 7> { using ZPZ = aerobus::zpz<463>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<460»; }; // NOLINT template<> struct ConwayPolynomial<463, 8> { using ZPZ = aerobus::zpz<463>; using type =
04652
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3414>, ZPZV<396>, ZPZV<39; }; //
             template<> struct ConwayPolynomial<463, 9> { using ZPZ = aerobus::zpz<463>; using type
04653
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<43>, ZPZV<43>, ZPZV<227>, ZPZV<460»;
        }; // NOLINT
04654
              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 =
        POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; }; // NOLINT
04656
             template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; }; // NOLINT
04657
```

```
04658
            template<> struct ConwayPolynomial<467, 5> { using ZPZ = aerobus::zpz<467>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
           template<> struct ConwayPolynomial<467, 6> { using ZPZ = aerobus::zpz<467>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<12>, ZPZV<62>, ZPZV<237>, ZPZV<2x; }; // NOLINT template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type
04660
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, 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»; }; //
       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<465»;</pre>
04662
       }; // NOLINT
04663
            template<> struct ConwayPolynomial<479, 1> { using ZPZ = aerobus::zpz<479>; using type =
       POLYV<ZPZV<1>, ZPZV<466»; }; // NOLINT
04664
            template<> struct ConwayPolynomial<479, 2> { using ZPZ = aerobus::zpz<479>; using type =
       POLYV<ZPZV<1>, ZPZV<474>, ZPZV<413»; }; // NOLINT template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
04665
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466»; };
                                                               // NOLINT
            template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
04666
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13»; }; // NOLINT
            template<> struct ConwayPolynomial<479, 5> { using ZPZ = aerobus::zpz<479>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<466»; }; // NOLINT
           template<> struct ConwayPolynomial<479, 6> { using ZPZ = aerobus::zpz<479>; using type =
04668
       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
04669
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<46*; }; //
           template<> struct ConwayPolynomial<479, 8> { using ZPZ = aerobus::zpz<479>; using type
04670
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<247>, ZPZV<440>, ZPZV<17>, ZPZV<13»; }; //
       NOLINT
04671
           template<> struct ConwayPolynomial<479, 9> { using ZPZ = aerobus::zpz<479>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<185>, ZPZV<466»; };
       // NOLINT
04672
            template<> struct ConwayPolynomial<487, 1> { using ZPZ = aerobus::zpz<487>; using type =
       POLYV<ZPZV<1>, ZPZV<484»; }; // NOLINT
04673
            template<> struct ConwayPolynomial<487, 2> { using ZPZ = aerobus::zpz<487>; using type =
       POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; }; // NOLINT
       template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; }; // NOLINT
04674
04675
            template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
04676
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<484»; }; // NOLINT template<> struct ConwayPolynomial<487, 6> { using ZPZ = aerobus::zpz<487>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<427>, ZPZV<185>, ZPZV<3»; }; // NOLINT
04677
            template<> struct ConwayPolynomial<487, 7> { using ZPZ = aerobus::zpz<487>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<484»; };
04679
           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»; }; //
       NOLINT
            template<> struct ConwayPolynomial<487, 9> { using ZPZ = aerobus::zpz<487>; using type =
04680
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<271>, ZPZV<2444>, ZPZV<484»;
       }; // NOLINT
04681
            template<> struct ConwayPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<489»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
04682
       POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<489»; }; // NOLINT template<>> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
04685
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT
04686
            template<> struct ConwayPolynomial<491, 6> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<369>, ZPZV<402>, ZPZV<125>, ZPZV<2»; }; // NOLINT
04687
           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<5>, ZPZV<5>, ZPZV<5>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 8> { using ZPZ = aerobus::zpz<491>; using type =
04688
       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<0>, ZPZV<149>, ZPZV<453>, ZPZV<489»;
       }; // NOLINT
04690
            template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
       POLYV<ZPZV<1>, ZPZV<492»; }; // NOLINT
            template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
04691
       POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
04692
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
04693
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
04694
       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 =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<407>, ZPZV<191>, ZPZV<78>, ZPZV<78>, // NOLINT
      template<> struct ConwayPolynomial<499, 7> { using ZPZ = aerobus::zpz<499>; using type =
POLYV<ZPZV<1>, 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 =
04696
04697
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<288>, ZPZV<309>, ZPZV<200>, ZPZV<7»; }; //
04698
           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
04699
           template<> struct ConwayPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<498»; }; // NOLINT
           template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
04700
      POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; }; // NOLINT
04701
           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 =
04702
      POLYV<ZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<325>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<503, 5> { using ZPZ = aerobus::zpz<503>; using type =
04703
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<498»; }; // NOLINT
04704
           template<> struct ConwayPolynomial<503, 6> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<380>, ZPZV<292>, ZPZV<255>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<503, 7> { using ZPZ = aerobus::zpz<503>; using type
04705
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<498»; }; // NOLINT
           template<> struct ConwayPolynomial<503, 8> { using ZPZ = aerobus::zpz<503>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<441>, ZPZV<203>, ZPZV<316>, ZPZV<5»; }; //
       NOLINT
04707
           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<35, ZPZV<158>, ZPZV<337>, ZPZV<498»;
       }; // NOLINT
            template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<507»; }; // NOLINT
04709
           template<> struct ConwayPolynomial<509, 2> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
04710
      POLYV<ZPZV<1>, ZPZV<3>, ZPZV<30, ZPZV<507*; }; // NOLINT template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
04711
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2»; }; // NOLINT
04712
          template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
      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 =
04713
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<507»; };
           template<> struct ConwayPolynomial<509, 8> { using ZPZ = aerobus::zpz<509>; using type =
04715
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<420>, ZPZV<473>, ZPZV<382>, ZPZV<2»; }; //
       NOLINT
           template<> struct ConwayPolynomial<509, 9> { using ZPZ = aerobus::zpz<509>; using type =
04716
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<314>, ZPZV<28>, ZPZV<507»;
       }; // NOLINT
04717
           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 =
04718
      POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
04719
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518»; }; // NOLINT template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
      04721
           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
           template<> struct ConwayPolynomial<521, 6> { using ZPZ = aerobus::zpz<521>; using type =
04722
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<315>, ZPZV<153>, ZPZV<280>, ZPZV<3»; }; // NOLINT
                                                          7> { using ZPZ = aerobus::zpz<521>; using type
04723
           template<> struct ConwayPolynomial<521,
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<518»; }; // NOLINT template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type =
04724
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<46>, ZPZV<407>, ZPZV<312>, ZPZV<3»; }; //
       NOLINT
           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
04726
           template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<521»; }; // NOLINT
           template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
04727
      POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
04728
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
04729
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2x; }; // NOLINT template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
04730
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<52); }; // NOLINT template<> struct ConwayPolynomial<523, 6> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<475>, ZPZV<475>, ZPZV<371>, ZPZV<2»; }; // NOLINT
      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»; }; // NOLII
template<> struct ConwayPolynomial<523, 8> { using ZPZ = aerobus::zpz<523>; using type =
04732
                                                                                                          // NOLINT
04733
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<518>, ZPZV<184>, ZPZV<380>, ZPZV<2»; }; //
      template<> struct ConwayPolynomial<523, 9> { using ZPZ = aerobus::zpz<523>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<342>, ZPZV<145>, ZPZV<145>, ZPZV<521»;
       }; // NOLINT
           template<> struct ConwayPolynomial<541, 1> { using ZPZ = aerobus::zpz<541>; using type =
04735
       POLYV<ZPZV<1>, ZPZV<539»; }; // NOLINT
```

```
04736
           template<> struct ConwayPolynomial<541, 2> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; }; // NOLINT
          template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539»; }; // NOLINT
template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2»; }; // NOLINT
04738
           template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<539»; }; // NOLINT
04740
          template<> struct ConwayPolynomial<541, 6> { using ZPZ = aerobus::zpz<541>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<239>, ZPZV<320>, ZPZV<69>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<541, 7> { using ZPZ = aerobus::zpz<541>; using type =
04741
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<539»; }; // NOLINT
           template<> struct ConwayPolynomial<541, 8> { using ZPZ = aerobus::zpz<541>; using type =
04742
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<376>, ZPZV<108>, ZPZV<113>, ZPZV<2»; }; //
      NOLINT
04743
          template<> struct ConwayPolynomial<541, 9> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<340>, ZPZV<318>, ZPZV<539»;
      }; // NOLINT
           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 =
      POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; }; // NOLINT
          template<> struct ConwayPolynomial<547, 3> { using ZPZ = aerobus::zpz<547>; using type =
04746
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 4> { using ZPZ = aerobus::zpz<547>; using type =
04747
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<334>, ZPZV<2»; }; // NOLINT
04748
          template<> struct ConwayPolynomial<547, 5> { using ZPZ = aerobus::zpz<547>; using type =
      04749
           template<> struct ConwayPolynomial<547, 6> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<547, 7> { using ZPZ = aerobus::zpz<547>; using type
04750
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<545»; };
           template<> struct ConwayPolynomial<547, 8> { using ZPZ = aerobus::zpz<547>;
04751
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2»; }; //
      NOLINT
           template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type =
04752
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<238>, ZPZV<263>, ZPZV<545»;
04753
           template<> struct ConwayPolynomial<557, 1> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<555»; }; // NOLINT
04754
          template<> struct ConwayPolynomial<557, 2> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
04755
      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
04757
          template<> struct ConwayPolynomial<557, 5> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<555»; }; // NOLINT
           template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type =
04758
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<202>, ZPZV<192>, ZPZV<253>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<557,
                                                     7> { using ZPZ = aerobus::zpz<55</pre>
                                                                                        7>; using type
      POLYV<ZPZV<1>, 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
04760
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<384>, ZPZV<113>, ZPZV<2»; }; //
      NOLINT
      template<> struct ConwayPolynomial<557, 9> { using ZPZ = aerobus::zpz<557>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<456>, ZPZV<436>, ZPZV<434>, ZPZV<555»;</pre>
04761
      }; // NOLINT
           template<> struct ConwayPolynomial<563, 1> { using ZPZ = aerobus::zpz<563>; using type =
04762
      POLYV<ZPZV<1>, ZPZV<561»; }; // NOLINT
          template<> struct ConwayPolynomial<563, 2> { using ZPZ = aerobus::zpz<563>; using type =
04763
      POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; }; // NOLINT
04764
           template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZV<561»; }; // NOLINT template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
04765
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<20>, ZPZV<399>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<563, 5> { using ZPZ = aerobus::zpz<563>; using type =
04766
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<561»; }; // NOLINT
          template<> struct ConwayPolynomial<563, 6> { using ZPZ = aerobus::zpz<563>; using type =
04767
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<303>, ZPZV<246>, ZPZV<246>, ZPZV<2); }; // NOLINT
           template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz<563>; using type =
04768
      04769
          template<> struct ConwayPolynomial<563, 8> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<176>, ZPZV<509>, ZPZV<2»; }; //
      NOLINT
           template<> struct ConwayPolynomial<563, 9> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<15>, ZPZV<19>, ZPZV<561»; };
04771
          template<> struct ConwayPolynomial<569, 1> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<566»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 2> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; };
                                                          // NOLINT
          template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
04774
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<569, 5> { using ZPZ = aerobus::zpz<569>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<566»; };
               template<> struct ConwayPolynomial<569, 6> { using ZPZ = aerobus::zpz<569>; using type =
        POLYV<2PZV<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<0>, ZPZV<5, ZPZV<566»; }; // NOLINT template<> struct ConwayPolynomial<569, 8> { using ZPZ = aerobus::zpz<569>; using type =
04778
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<527>, ZPZV<173>, ZPZV<241>, ZPZV<241>, ZPZV<3»; }; //
04779
              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
04780
               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 =
        POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
04782
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3»; }; // NOLINT
04783
              template<> struct ConwayPolynomial<571, 5> { using ZPZ = aerobus::zpz<571>; using type =
04784
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<568»; }; // NOLINT
04785
              \texttt{template<>} \texttt{struct ConwayPolynomial<571, 6> \{ \texttt{using ZPZ = aerobus::zpz<571>; using type = 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 20
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<221>, ZPZV<295>, ZPZV<33>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<571, 7> { using ZPZ = aerobus::zpz<571>; using type =
04786
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<568»; }; // NOLINT
               template<> struct ConwayPolynomial<571, 8> { using ZPZ = aerobus::zpz<571; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<363>, ZPZV<119>, ZPZV<371>, ZPZV<37; }; //
        NOLINT
        template<> struct ConwayPolynomial<571, 9> { using ZPZ = aerobus::zpz<571>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<545>, ZPZV<5179>, ZPZV<568»;
04788
        }; // NOLINT
04789
               template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
        POLYV<ZPZV<1>, ZPZV<572»; }; // NOLINT
04790
              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 =
04791
        POLYV<ZPZV<1>, ZPZV<2>, 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 =
04793
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<572»; }; // NOLINT
               template<> struct ConwayPolynomial<577, 6> { using ZPZ = aerobus::zpz<577>; using type =
04794
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<450>, ZPZV<25>, ZPZV<283>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<577, 7> { using ZPZ = aerobus::zpz<577>; using type
04795
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<572»; }; // NOLINT
04796
              template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<32; }; //
         NOLINT
04797
              template<> struct ConwayPolynomial<577, 9> { using ZPZ = aerobus::zpz<577>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<576>, ZPZV<444>, ZPZV<572»;
        }; // NOLINT template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
04798
        POLYY<ZPZV<1>, ZPZV<585»; }; // NOLINT template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
04799
        POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<587, 3> { using ZPZ = aerobus::zpz<587>; using type =
04800
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585»; }; // NOLINT
              template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
04801
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<16>, ZPZY<444>, ZPZY<2w; }; // NOLINT template<> struct ConwayPolynomial<587, 5> { using ZPZ = aerobus::zpz<587>; using type =
04802
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<585»; }; // NOLINT
        template<> struct ConwayPolynomial<587, 6> { using ZPZ = aerobus::zpz<587>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<204>, ZPZV<204>, ZPZV<2121>, ZPZV<226>, ZPZV<2»; }; // NOLINT
04803
               template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<585»; };
04805
              template<> struct ConwayPolynomial<587, 8> { using ZPZ = aerobus::zpz<587>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<492>, ZPZV<44>, ZPZV<91>, ZPZV<91>, ZPZV<2»; };
         NOLINT
              template<> struct ConwayPolynomial<587, 9> { using ZPZ = aerobus::zpz<587>; using type =
04806
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<55>, ZPZV<585»;
         }; // NOLINT
04807
              template<> struct ConwayPolynomial<593, 1> { using ZPZ = aerobus::zpz<593>; using type =
        POLYV<ZPZV<1>, ZPZV<590»; }; // NOLINT
               template<> struct ConwayPolynomial<593, 2> { using ZPZ = aerobus::zpz<593>; using type =
04808
        POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<593, 3> { using ZPZ = aerobus::zpz<593>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<590»; }; // NOLINT template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
04810
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<593, 5> { using ZPZ = aerobus::zpz<593>; using type =
04811
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<590»; }; // NOLINT
04812
               template<> struct ConwayPolynomial<593, 6> { using ZPZ = aerobus::zpz<593>; using type =
        POLYV<ZPZV<1>, ZPZV<3>, ZPZV<2>, ZPZV<345>, ZPZV<478>, ZPZV<3>; ); // NOLINT template<> struct ConwayPolynomial<593, 7> { using ZPZ = aerobus::zpz<593>; using type =
04813
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>0»; }; // NOLINT template<> struct ConwayPolynomial<593, 8> { using ZPZ = aerobus::zpz<593>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<29>, ZPZV<350>, ZPZV<291>, ZPZV<495>, ZPZV<3»; }; //
04814
```

```
NOLINT
         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<23>, ZPZV<223>, ZPZV<523>, ZPZV<590»;
04815
         }; // NOLINT
04816
               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
04818
               template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<592»; }; // NOLINT template<> struct ConwayPolynomial<599, 4> { using ZPZ = aerobus::zpz<599>; using type =
04819
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<599, 5> { using ZPZ = aerobus::zpz<599>; using type =
04820
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<592»; }; // NOLINT
04821
               template<> struct ConwayPolynomial<599, 6> { using ZPZ = aerobus::zpz<599>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type =
04822
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592»; }; // NOLINT
               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»; }; //
04824
               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<0>, ZPZV<3>, ZPZV<114>, ZPZV<98>, ZPZV<592»;
         }; // NOLINT
04825
               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 =
04826
         POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
04827
               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 =
04828
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<601, 5> { using ZPZ = aerobus::zpz<601>; using type =
04829
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<594»; }; // NOLINT
04830
               template<> struct ConwayPolynomial<601, 6> { using ZPZ = aerobus::zpz<601>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<440>, ZPZV<49>, ZPZV<7>; }; // NOLINT template<> struct ConwayPolynomial<601, 7> { using ZPZ = aerobus::zpz<601>; using type :
04831
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<594»; }; //
               template<> struct ConwayPolynomial<601, 8> { using ZPZ = aerobus::zpz<601>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<550>, ZPZV<241>, ZPZV<490>, ZPZV<7»; }; //
         NOLINT
         template<> struct ConwayPolynomial<601, 9> { using ZPZ = aerobus::zpz<601>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<487>, ZPZV<487>, ZPZV<590>, ZPZV<594»;</pre>
04833
         }; // NOLINT
               template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<604»; }; // NOLINT
04835
               template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
04836
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<604»; }; // NOLINT template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
04837
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3»; }; // NOLINT
04838
               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 template<> struct ConwayPolynomial<607, 6> { using ZPZ = aerobus::zpz<607>; using type =
04839
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<478>, ZPZV<48>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               template<> struct ConwayPolynomial<607, 8> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<468>, ZPZV<35>, ZPZV<449>, ZPZV<3»; }; //
         NOLINT
         template<> struct ConwayPolynomial<607, 9> { using ZPZ = aerobus::zpz<607>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<444>, ZPZV<444>, ZPZV<129>, ZPZV<604;
04842
         }; // NOLINT
               template<> struct ConwayPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
04843
         POLYV<ZPZV<1>, ZPZV<611»; }; // NOLINT
04844
               template<> struct ConwayPolynomial<613, 2> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
04845
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<65, ZPZV<611»; }; // NOLINT template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
04846
          \verb"POLYV<ZPZV<1>, \ \verb"ZPZV<0>, \ \verb"ZPZV<12>, \ \verb"ZPZV<333>, \ \verb"ZPZV<2"; \ \verb"}; \ \ // \ \verb"NOLINT" 
               template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
04847
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<611»; }; // NOLINT
               template<> struct ConwayPolynomial<613, 6> { using ZPZ = aerobus::zpz<613>; using type =
04848
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<609>, ZPZV<595>, ZPZV<601>, ZPZV<2»; }; // NOLINI
               template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type
04849
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<61|m; }; // NOLINT template<> struct ConwayPolynomial<613, 8> { using ZPZ = aerobus::zpz<613>; using type =
04850
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<489>, ZPZV<57>, ZPZV<539>, ZPZV<539>, ZPZV<59*, };
         NOLINT
04851
               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<5), ZPZV<513>, ZPZV<516, ZPZV<611»;
         }; // NOLINT
04852
               POLYV<ZPZV<1>, ZPZV<614»; }; // NOLINT
               template<> struct ConwayPolynomial<617, 2> { using ZPZ = aerobus::zpz<617>; using type =
04853
```

```
POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; };
               template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<614»; }; // NOLINT template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type =
04855
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<617, 5> { using ZPZ = aerobus::zpz<617>; using type =
04856
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<614»; }; // NOLINT
04857
               template<> struct ConwayPolynomial<617, 6> { using ZPZ = aerobus::zpz<617>; using type =
          \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 318>, \quad \texttt{ZPZV} < 595>, \quad \texttt{ZPZV} < 310>, \quad \texttt{ZPZV} < 3»; \quad \texttt{}; \quad \texttt{}// \quad \texttt{NOLINT} 
04858
              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<0>, ZPZV<614»; }; // NOLINT template<> struct ConwayPolynomial<617, 8> { using ZPZ = aerobus::zpz<617>; using type =
04859
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<519>, ZPZV<501>, ZPZV<155>, ZPZV<3»; }; //
04860
              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<0>, ZPZV<15>, ZPZV<388>, ZPZV<543>, ZPZV<614»;
         }; // NOLINT
04861
               template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<617»; }; // NOLINT
               template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; }; // NOLINT
04863
               template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617»; }; // NOLINT
template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492>, ZPZV<2»; }; // NOLINT
04864
               template<> struct ConwayPolynomial<619, 5> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<617»; }; // NOLINT
04866
               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 =
04867
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0), ZPZV<0>, ZPZV<0
         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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<579>, ZPZV<510>, ZPZV<617»;</pre>
04869
         }; // NOLINT
               template<> struct ConwayPolynomial<631, 1> { using ZPZ = aerobus::zpz<631>; using type =
         POLYV<ZPZV<1>, ZPZV<628»; }; // NOLINT
              template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
        POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<631, 3> { using ZPZ = aerobus::zpz<631>; using type =
04872
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT template<> struct ConwayPolynomial<631, 4> { using ZPZ = aerobus::zpz<631>; using type =
04873
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<3»; }; // NOLINT
04874
               template<> struct ConwayPolynomial<631, 5> { using ZPZ = aerobus::zpz<631>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
04875
              template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<516>, ZPZV<541>, ZPZV<106>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<631, 7> { using ZPZ = aerobus::zpz<631>; using type =
04876
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<628»; };
               template<> struct ConwayPolynomial<631, 8> { using ZPZ = aerobus::zpz<631>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<3»; }; //
04878
              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
   template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
04879
         POLYV<ZPZV<1>, ZPZV<638»; }; // NOLINT
               template<> struct ConwayPolynomial<641, 2> { using ZPZ = aerobus::zpz<641>; using type =
04880
         POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
04881
         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<629>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<641, 5> { using ZPZ = aerobus::zpz<641>; using type =
04883
         template<> struct ConwayPolynomial<641, 6> { using ZPZ = aerobus::zpz<641>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<294>, ZPZV<3»; }; // NOLINT
04884
               template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<638»; }; // NOLIN template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>; using type =
04886
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<3»; }; //
         NOLINT
              template<> struct ConwayPolynomial<641, 9> { using ZPZ = aerobus::zpz<641>; using type
04887
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<668>, ZPZV<141>, ZPZV<638»;
         }; // NOLINT
04888
               template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
         POLYV<ZPZV<1>, ZPZV<632»; }; // NOLINT
               template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
04889
         POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11»; }; // NOLINT
               template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<632»; }; // NOLINT
              template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
04891
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<600>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<643, 5> { using ZPZ = aerobus::zpz<643>; using type =
04892
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<632»; }; // NOLINT
```

```
04893
                   template<> struct ConwayPolynomial<643, 6> { using ZPZ = aerobus::zpz<643>; using type =
           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 =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<632»; }; // NOLINT template<> struct ConwayPolynomial<643, 8> { using ZPZ = aerobus::zpz<643>; using type =
04895
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<573>, ZPZV<569>, ZPZV<11»; }; //
           NOLINT
                   template<> struct ConwayPolynomial<643, 9> { using ZPZ = aerobus::zpz<643>; using type
04896
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<59, ZPZV<591>, ZPZV<475>, ZPZV<632»;
           }; // NOLINT
04897
                  template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
           POLYV<ZPZV<1>, ZPZV<642»; }; // NOLINT
                  template<> struct ConwayPolynomial<647, 2> { using ZPZ = aerobus::zpz<647>; using type =
04898
           POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5»; }; // NOLINT
04899
                  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 =
04900
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<643>, ZPZV<645, ZPZV<543, ZPZV<645, ZPZV<647, Z
           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 =
04902
           POLYV<2PZV<1>, 2PZV<0>, 2PZV<5>, 2PZV<308>, ZPZV<385>, ZPZV<642>, ZPZV<5»; }; // NOLINT
04903
                  template<> struct ConwayPolynomial<647, 7> { using ZPZ = aerobus::zpz<647>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04904
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<603>, ZPZV<259>, ZPZV<271>, ZPZV<5»; }; //
           template<> struct ConwayPolynomial<647, 9> { using ZPZ = aerobus::zpz<647>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<561>, ZPZV<561>, ZPZV<123>, ZPZV<642»;</pre>
04905
           }; // NOLINT
                  template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
04906
           POLYV<ZPZV<1>, ZPZV<651»; }; // NOLINT
                  template<> struct ConwayPolynomial<653, 2> { using ZPZ = aerobus::zpz<653>; using type =
04907
           POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2»; }; // NOLINT
04908
                   template<> struct ConwayPolynomial<653, 3> { using ZPZ = aerobus::zpz<653>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<651»; }; // NOLINT template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2»; }; // NOLINT
04909
04910
                   template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651»; }; // NOLINT
04911
                  template<> struct ConwayPolynomial<653, 6> { using ZPZ = aerobus::zpz<653>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<45>, ZPZV<220>, ZPZV<242>, ZPZV<2*; }; // NOLINT
                  template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type
04912
           POLYV<ZPZV<1>, 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
           POLYV<2PZV<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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65), ZPZV<665), ZPZV<651»;</pre>
04914
           }; // NOLINT
04915
                   template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
           POLYV<ZPZV<1>, ZPZV<657»; }; // NOLINT
04916
                   template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
           POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
04917
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<657»; }; // NOLINT template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2»; }; // NOLINT
                   template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
04919
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<657»; }; // NOLINT
                  template<> struct ConwayPolynomial<659, 6> { using ZPZ = aerobus::zpz<659>; using type =
04920
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<371>, ZPZV<223>, ZPZV<223>, ZPZV<223>; // NOLINT template<> struct ConwayPolynomial<659, 7> { using ZPZ = aerobus::zpz<659>; using type
04921
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<657»; }; //
04922
                  template<> struct ConwayPolynomial<659, 8> { using ZPZ = aerobus::zpz<659>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<358>, ZPZV<246>, ZPZV<90>, ZPZV<2»; };
           NOLINT
04923
                  template<> struct ConwayPolynomial<659, 9> { using ZPZ = aerobus::zpz<659>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592>, ZPZV<592>, ZPZV<657»;
           }; // NOLINT
04924
                  template<> struct ConwayPolynomial<661, 1> { using ZPZ = aerobus::zpz<661>; using type =
           POLYV<ZPZV<1>, ZPZV<659»; }; // NOLINT
                  template<> struct ConwayPolynomial<661, 2> { using ZPZ = aerobus::zpz<661>; using type =
04925
           POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<659»; }; // NOLINT
04926
                  template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type =
04927
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
04928
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<659»; }; // NOLINT
          template<> struct ConwayPolynomial<661, 6> { using ZPZ = aerobus::2pz<661>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<456>, ZPZV<382>, ZPZV<2»; }; // NOLINT
04929
                   template<> struct ConwayPolynomial<661, 7> { using ZPZ = aerobus::zpz<661>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<659»; };
04931
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<612>, ZPZV<285>, ZPZV<72>, ZPZV<2»; }; //
           NOLTNT
```

```
template<> struct ConwayPolynomial<661, 9> { using ZPZ = aerobus::zpz<661>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<389>, ZPZV<220>, ZPZV<659»;
       }; // NOLINT
04933
           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 =
04934
      POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5»; }; // NOLINT
            template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
04935
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
04936
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<673, 5> { using ZPZ = aerobus::zpz<673>; using type =
04937
      POLYY<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
04939
           template<> struct ConwayPolynomial<673, 7> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<668»; }; // NOLINT
           template<> struct ConwayPolynomial<673, 8> { using ZPZ = aerobus::zpz<673>; using type =
04940
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<587>, ZPZV<302>, ZPZV<5»; }; //
      template<> struct ConwayPolynomial<673, 9> { using ZPZ = aerobus::zpz<673>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<347>, ZPZV<347>, ZPZV<553>, ZPZV<668»;
04941
       }; // NOLINT
            template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
04942
      POLYV<ZPZV<1>, ZPZV<675»; }; // NOLINT
            template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<672>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<677, 3> { using ZPZ = aerobus::zpz<677>; using type =
04944
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<675»; }; // NOLINT
template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<677, 5> { using ZPZ = aerobus::zpz<677>; using type =
04945
04946
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<675»; }; // NOLINT
04947
           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 =
04948
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<675»; }; // NOLINT template<> struct ConwayPolynomial<677, 8> { using ZPZ = aerobus::zpz<677>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<619>, ZPZV<152>, ZPZV<2*; }; //
04950
           template<> struct ConwayPolynomial<677, 9> { using ZPZ = aerobus::zpz<677>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<504>, ZPZV<504>, ZPZV<404, ZPZV<675»;
       }; // NOLINT
04951
            template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<678»; }; // NOLINT
04952
            template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; }; // NOLINT
04953
           template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<678»; }; // NOLINT
           template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type =
04954
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5»; }; // NOLINT
            template<> struct ConwayPolynomial<683, 5> { using ZPZ = aerobus::zpz<683>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<678»; }; // NOLINT
04956
           template<> struct ConwayPolynomial<683, 6> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<644>, ZPZV<434>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<683, 7> { using ZPZ = aerobus::zpz<683>; using type
04957
       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 =
04958
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<383>, ZPZV<184>, ZPZV<65>, ZPZV<5»; };
       NOLINT
           template<> struct ConwayPolynomial<683, 9> { using ZPZ = aerobus::zpz<683>; using type =
04959
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<678»;
       }; // NOLINT
04960
            template<> struct ConwayPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<688»; }; // NOLINT
04961
           template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
       POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; }; // NOLINT
04962
           template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<688»; }; // NOLINT template<> struct ConwayPolynomial<691, 4> { using ZPZ = aerobus::zpz<691>; using type =
04963
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<691, 5> { using ZPZ = aerobus::zpz<691>; using type =
04964
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; }; // NOLINT template<> struct ConwayPolynomial<691, 6> { using ZPZ = aerobus::zpz<691>; using type =
04965
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<262>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; };
04967
           template<> struct ConwayPolynomial<691, 8> { using ZPZ = aerobus::zpz<691>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<356>, ZPZV<425>, ZPZV<321>, ZPZV<3»; }; //
      NOLINT
           template<> struct ConwayPolynomial<691, 9> { using ZPZ = aerobus::zpz<691>; using type =
04968
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<55, ZPZV<556>, ZPZV<443>, ZPZV<688»;
       }; // NOLINT
04969
           template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<699»; }; // NOLINT template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
04970
       POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; }; // NOLINT
```

```
04971
            template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; }; // NOLINT template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
04972
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<379, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
04973
       POLYV<ZPZV<1>, ZPZV<0>, 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»; }; // NOLIN
04975
           template<> struct ConwayPolynomial<701, 7> { using ZPZ = aerobus::zpz<701>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<699»; };
04976
           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<2*; }; //
       NOLINT
            template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<459>, ZPZV<373>, ZPZV<699»;
       }; // NOLINT
           template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
04978
       POLYV<ZPZV<1>, ZPZV<707»; }; // NOLINT
            template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
       POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707»; }; // NOLINT template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
04981
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<709, 5> { using ZPZ = aerobus::zpz<709>; using type =
04982
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<707»; }; // NOLINT
           template<> struct ConwayPolynomial<709, 6> { using ZPZ = aerobus::zpz<709>; using type =
04983
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<669>, ZPZV<514>, ZPZV<295>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<709, 7> { using ZPZ = aerobus::zpz<709>; using type
04984
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<707»; }; // NOLINT
           template<> struct ConwayPolynomial<709, 8> { using ZPZ = aerobus::zpz<709>; using type =
04985
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<689>, ZPZV<233>, ZPZV<79>, ZPZV<2»; };
           template<> struct ConwayPolynomial<709, 9> { using ZPZ = aerobus::zpz<709>; using type =
04986
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<257>, ZPZV<171>, ZPZV<707»;
       }; // NOLINT
       template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type = POLYV<ZPZV<1>, ZPZV<708»; }; // NOLINT
04987
            template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
       POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; }; // NOLINT
04989
           template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<602>, ZPZV<11»; }; // NOLINT
04990
            template<> struct ConwayPolynomial<719, 5> { using ZPZ = aerobus::zpz<719>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
04992
           template<> struct ConwayPolynomial<719, 6> { using ZPZ = aerobus::zpz<719>; using type =
        \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 533>, \ \texttt{ZPZV} < 591>, \ \texttt{ZPZV} < 182>, \ \texttt{ZPZV} < 11»; \ \}; \ \ // \ \ \texttt{NOLINT} 
           template<> struct ConwayPolynomial<719, 7> { using ZPZ = aerobus::zpz<719>; using type
04993
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<708»; }; // NOLINT
            template<> struct ConwayPolynomial<719, 8> { using ZPZ = aerobus::zpz<719>; using type
04994
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<714>, ZPZV<362>, ZPZV<244>, ZPZV<211»; }; //
       NOLINT
       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<28>, ZPZV<288>, ZPZV<260>, ZPZV<708»;</pre>
04995
       }; // NOLINT
            template<> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
       POLYV<ZPZV<1>, ZPZV<722»; }; // NOLINT
            template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
04997
       POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
04998
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<722»; }; // NOLINT template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
04999
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5»; }; // NOLINT
05000
           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 =
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 =
05001
05002
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<722»; }; //
           template<> struct ConwayPolynomial<727, 8> { using ZPZ = aerobus::zpz<727>; using type =
05003
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<639>, ZPZV<671>, ZPZV<368>, ZPZV<5»; }; //
       NOLINT
           template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type =
05004
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<573>, ZPZV<502>, ZPZV<722»;
       }; // NOLINT
05005
            template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
       POLYV<ZPZV<1>, ZPZV<727»; }; // NOLINT
           template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
05006
       POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; }; // NOLINT
           template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
05007
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727»; }; // NOLINT
            template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<6»; }; // NOLINT
05009
           template<> struct ConwayPolynomial<733, 5> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<727»; }; // NOLINT
05010
           template<> struct ConwayPolynomial<733, 6> { using ZPZ = aerobus::zpz<733>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<549>, ZPZV<151>, ZPZV<6»; };
                  template<> struct ConwayPolynomial<733, 7> { using ZPZ = aerobus::zpz<733>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<727»; }; // NOLINT
05012
                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*; }; //
          NOLTNT
                 template<> struct ConwayPolynomial<733, 9> { using ZPZ = aerobus::zpz<733>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<337>, ZPZV<6>, ZPZV<727»; };
           // NOLINT
05014
                 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 =
05015
          POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; }; // NOLINT
                 template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
05017
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<3»; }; // NOLINT
                 template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
05018
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<736»; }; // NOLINT
                 template<> struct ConwayPolynomial<739, 6> { using ZPZ = aerobus::zpz<739>; using type =
05019
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<422>, ZPZV<447>, ZPZV<625>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<739, 7> { using ZPZ = aerobus::zpz<739>; using type =
05020
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3, ZPZV<4>, ZPZV<36»; }; // NOLINT template<> struct ConwayPolynomial<739, 8> { using ZPZ = aerobus::zpz<739>; using type =
05021
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<401>, ZPZV<169>, ZPZV<25>, ZPZV<3»; };
                 template<> struct ConwayPolynomial<739, 9> { using ZPZ = aerobus::zpz<739>; using type =
05022
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<616>, ZPZV<81>, ZPZV<736»;
          }; // NOLINT
05023
                 template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
          POLYV<ZPZV<1>, ZPZV<738»; }; // NOLINT
05024
                  template<> struct ConwayPolynomial<743, 2> { using ZPZ = aerobus::zpz<743>; using type =
          POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; }; // NOLINT
05025
                 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 =
05026
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<425>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<743, 5> { using ZPZ = aerobus::zpz<743>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT
                 template<> struct ConwayPolynomial<743, 6> { using ZPZ = aerobus::zpz<743>; using type =
05028
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<236>, ZPZV<471>, ZPZV<88>, ZPZV<5»; }; // NOLINT
                 template<> struct ConwayPolynomial<743, 7> { using ZPZ = aerobus::zpz<743>; using type =
05029
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT
                 template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
05030
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<279>, ZPZV<588>, ZPZV<5»; }; //
          NOLINT
05031
                 template<> struct ConwayPolynomial<743, 9> { using ZPZ = aerobus::zpz<743>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<5>, ZPZV<5-, ZPZV<5
          }; // NOLINT
05032
                  template<> struct ConwavPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
          POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
                  template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
          POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT
                 template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
05034
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; }; // NOLINT template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
05035
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<525>, ZPZV<3»; };
                                                                                                                // NOLINT
                 template<> struct ConwayPolynomial<751, 5> { using ZPZ = aerobus::zpz<751>; using type =
05036
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT
05037
                  template<> struct ConwayPolynomial<751, 6> { using ZPZ = aerobus::zpz<751>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<298>, ZPZV<633>, ZPZV<539>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<751, 7> { using ZPZ = aerobus::zpz<751>; using type
05038
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<748»; }; // NOLINT
                 template<> struct ConwayPolynomial<751, 8> { using ZPZ = aerobus::zpz<751>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<741>, ZPZV<243>, ZPZV<672>, ZPZV<672>, ZPZV<3»; }; //
          NOLINT
          template<> struct ConwayPolynomial<751, 9> { using ZPZ = aerobus::zpz<751>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<70>, ZPZV<70>, ZPZV<70>, ZPZV<489>, ZPZV<70>
05040
          }; // NOLINT
05041
                  template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
          POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
05042
                 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 =
05043
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<6>, ZPZY<755»; }; // NOLINT template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; }; // NOLINT
          template<> struct ConwayPolynomial<757, 5> { using ZPZ = aerobus::zpz<757>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<755»; }; // NOLINT</pre>
05045
05046
                 template<> struct ConwayPolynomial<757, 6> { using ZPZ = aerobus::zpz<757>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<753>, ZPZV<739>, ZPZV<745>, ZPZV<2»; }; // NOLINT
                                                                                                                                                   7>; using type
                 template<> struct ConwayPolynomial<757,
                                                                                        7> { using ZPZ = aerobus::zpz<75
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<0>, ZPZV<0>
05048
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<110>, ZPZV<509>, ZPZV<2»; }; //
          NOLINT
05049
                 template<> struct ConwayPolynomial<757, 9> { using ZPZ = aerobus::zpz<757>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<88>, ZPZV<688>, ZPZV<702>, ZPZV<755»;
05050
             template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
        POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
05051
              template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
        POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; }; // NOLINT
              template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755»; }; // NOLINT
             template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type =
05053
        template<> struct ConwayPolynomial<761, 5> { using ZPZ = aerobus::zpz<761>; using type =
05054
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT
              template<> struct ConwayPolynomial<761, 6> { using ZPZ = aerobus::zpz<761>; using type =
05055
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<634>, ZPZV<597>, ZPZV<155>, ZPZV<6»; }; // NOLIN
05056
             template<> struct ConwayPolynomial<761, 7> { using ZPZ = aerobus::zpz<761>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<755»; }; // NOLIN template<> struct ConwayPolynomial<761, 8> { using ZPZ = aerobus::zpz<761>; using type =
05057
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0), ZPZV<11>, ZPZV<603>, ZPZV<144>, ZPZV<540>, ZPZV<6»; }; //
05058
             template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<317>, ZPZV<571>, ZPZV<755»;
        }; // NOLINT
05059
             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 =
05060
        POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; }; // NOLINT
              template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
05061
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; }; // NOLINT
              template<> struct ConwayPolynomial<769, 4> { using ZPZ = aerobus::zpz<769>; using type =
05062
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<769, 5> { using ZPZ = aerobus::zpz<769>; using type =
05063
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<758»; }; // NOLINT
              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»; }; // NOLINI
05065
              template<> struct ConwayPolynomial<769, 7> { using ZPZ = aerobus::zpz<769>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<758»; }; // NOLINT
             template<> struct ConwayPolynomial<769, 8> { using ZPZ = aerobus::zpz<769>; using type =
05066
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<560>, ZPZV<574>, ZPZV<632>, ZPZV<11»; }; //
             template<> struct ConwayPolynomial<769, 9> { using ZPZ = aerobus::zpz<769>; using type =
05067
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<623>, ZPZV<751>, ZPZV<758»;
        }; // NOLINT
              template<> struct ConwayPolynomial<773, 1> { using ZPZ = aerobus::zpz<773>; using type =
05068
        POLYV<ZPZV<1>, ZPZV<771»; }; // NOLINT
              template<> struct ConwayPolynomial<773, 2> { using ZPZ = aerobus::zpz<773>; using type =
        POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»; }; // NOLINT
05070
             template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<771»; }; // NOLINT template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
05071
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<773, 5> { using ZPZ = aerobus::zpz<773>; using type =
05072
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<771»; }; // NOLINT
05073
              template<> struct ConwayPolynomial<773, 6> { using ZPZ = aerobus::zpz<773>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<51>, ZPZV<581>, ZPZV<581>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<773, 7> { using ZPZ = aerobus::zpz<773>; using type =
05074
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<771»; }; // NOLINT template<> struct ConwayPolynomial<773, 8> { using ZPZ = aerobus::zpz<773>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<484>, ZPZV<94>, ZPZV<693>, ZPZV<2»; };
05076
              template<> struct ConwayPolynomial<773, 9> { using ZPZ = aerobus::zpz<773>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<216>, ZPZV<574>, ZPZV<771»;
        }; // NOLINT
05077
              template<> struct ConwayPolynomial<787, 1> { using ZPZ = aerobus::zpz<787>; using type =
        POLYV<ZPZV<1>, ZPZV<785»; }; // NOLINT
05078
             template<> struct ConwayPolynomial<787, 2> { using ZPZ = aerobus::zpz<787>; using type =
        POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
05079
              template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785»; }; // NOLINT
             template<> struct ConwayPolynomial</ri>
05080
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2»; };
                                                                                         // NOLINT
              template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
05081
         \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<9>, \verb"ZPZV<785"; $$ // \verb"NOLINT" | NOLINT" | NO
05082
              template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<98>, ZPZV<512>, ZPZV<606>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<787, 7> { using ZPZ = aerobus::zpz<787>; using type
05083
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<785»; }; // NOLINT
             template<> struct ConwayPolynomial<787, 8> { using ZPZ = aerobus::zpz<787>; using type =
05084
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<612>, ZPZV<26>, ZPZV<715>, ZPZV<22»; };
        NOLINT
        template<> struct ConwayPolynomial<787, 9> { using ZPZ = aerobus::zpz<787>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<573>, ZPZV<785»;</pre>
05085
        }; // NOLINT
    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 =
05087
        POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»; }; // NOLINT
             template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
05088
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<795»; };
                                                               // NOLINT
       template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<717>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<797, 5> { using ZPZ = aerobus::zpz<797>; using type =
05090
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<795»; }; // NOLINT
       template<> struct ConwayPolynomial<797, 6> { using ZPZ = aerobus::zpz<797>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<657>, ZPZV<396>, ZPZV<71>, ZPZV<2»; }; // NOLINT
05091
            template<> struct ConwayPolynomial<797, 7> { using ZPZ = aerobus::zpz<797>; using type =
05092
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type =
05093
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2»; }; //
       NOLINT
           template<> struct ConwayPolynomial<797, 9> { using ZPZ = aerobus::zpz<797>; using type =
05094
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<240>, ZPZV<599>, ZPZV<795»;
       }; // NOLINT
05095
            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 =
05096
       POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
05097
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<806»; }; // NOLINT template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
05098
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<809, 5> { using ZPZ = aerobus::zpz<809>; using type =
05099
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; }; // NOLINT
            template<> struct ConwayPolynomial<809, 6> { using ZPZ = aerobus::zpz<809>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<562>, ZPZV<75>, ZPZV<43>, ZPZV<3»; }; // NOLINT
05101
           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<806»; }; // NOLINT
05102
           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
            template<> struct ConwayPolynomial<809, 9> { using ZPZ = aerobus::zpz<809>; using type =
05103
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<727>, ZPZV<806»;
       }; // NOLINT
05104
            template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
       POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
            template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
       POLYV<ZPZV<1>, ZPZV<806>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<811, 3> { using ZPZ = aerobus::zpz<811>; using type =
05106
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<818»; }; // NOLINT
template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<811, 5> { using ZPZ = aerobus::zpz<811>; using type =
05107
05108
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<808»; }; // NOLINT
05109
            template<> struct ConwayPolynomial<811, 6> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<780>, ZPZV<755>, ZPZV<307>, ZPZV<3»; }; // NOLINT
05110
           template<> struct ConwayPolynomial<811, 7> { using ZPZ = aerobus::zpz<811>; 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>; Using ZPZ = aerobus::zpz<811>; using type =
05111
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<525>, ZPZV<3»; }; //
05112
           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<10>, ZPZV<382>, ZPZV<380>, ZPZV<808»;
       }; // NOLINT
05113
            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 =
05114
       POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT
05115
            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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<662>, ZPZV<2»; }; // NOLINT
05116
            template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<819»; }; // NOLINT
05118
           template<> struct ConwayPolynomial<821, 6> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<160>, ZPZV<130>, ZPZV<803>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type
05119
       POLYV<ZPZV<1>, 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 =
       POLYV<ZPZV<1>, ZPZV<0>, ZPŽV<0>, ZPZV<0>, ZPZV<5>, ZPZV<556>, ZPZV<589>, ZPŽV<2»; }; //
       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<65), ZPZV<650>, ZPZV<557>, ZPZV<819»;</pre>
05121
       }; // NOLINT
05122
            template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
       POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
05123
            template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
05124
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
05126
      05127
       template<> struct ConwayPolynomial<823, 6> { using ZPZ = aerobus::zpz<823>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<822>, ZPZV<616>, ZPZV<744>, ZPZV<3»; }; // NOLINT</pre>
```

```
template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<820»; }; // NOLINT
             template<> struct ConwayPolynomial<823, 8> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<451>, ZPZV<437>, ZPZV<31>, ZPZV<3»; }; //
        NOLINT
              template<> struct ConwayPolynomial<823, 9> { using ZPZ = aerobus::zpz<823>; using type =
05130
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<740>, ZPZV<609, ZPZV<820»;
        }; // NOLINT
05131
              template<> struct ConwayPolynomial<827, 1> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<825»; }; // NOLINT
              template<> struct ConwayPolynomial<827, 2> { using ZPZ = aerobus::zpz<827>; using type =
05132
        POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
05133
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825»; }; // NOLINT
05134
             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 =
05135
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>; JPZV<3>, ZPZV<825; }; // NOLINT template<> struct ConwayPolynomial<827, 6> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<2PZV<1>, 2PZV<0>, ZPZV<9>, ZPZV<685>, ZPZV<601>, ZPZV<691>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<827, 7> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<825»; }; // NOLINT
05138
              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<32>; };
        NOLINT
              template<> struct ConwayPolynomial<827, 9> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<372>, ZPZV<825»;
        }; // NOLINT
05140
              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 =
        POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<827»; }; // NOLINT
              template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
05143
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<604>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<829, 5> { using ZPZ = aerobus::zpz<829>; using type =
05144
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<827»; }; // NOLINT
              template<> struct ConwayPolynomial<829, 6> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<341>, ZPZV<341>, ZPZV<817>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type =
05146
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<827»; }; // NOLINT template<> struct ConwayPolynomial<829, 8> { using ZPZ = aerobus::zpz<829>; using type =
05147
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<468>, ZPZV<241>, ZPZV<138>, ZPZV<2»; }; //
05148
              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
05149
              template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<828»; }; // NOLINT
05150
              template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; }; // NOLINT
05151
              template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; }; // NOLINT template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
05152
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<828»; }; // NOLINT
              template<> struct ConwayPolynomial<839, 6> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<370>, ZPZV<537>, ZPZV<23>, ZPZV<11»; }; // NOLINT
05155
              template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<828»; }; // NOLINT
              template<> struct ConwayPolynomial<839, 8> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<553>, ZPZV<779>, ZPZV<329>, ZPZV<11»; }; //
        template<> struct ConwayPolynomial<839, 9> { using ZPZ = aerobus::zpz<839>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<349>, ZPZV<206>, ZPZV<828*;</pre>
05157
        }; // NOLINT
              template<> struct ConwayPolynomial<853, 1> { using ZPZ = aerobus::zpz<853>; using type =
05158
        POLYV<ZPZV<1>, ZPZV<851»; }; // NOLINT
              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 =
0.5160
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
05161
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
05162
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<851»; }; // NOLINT
05163
              template<> struct ConwayPolynomial<853, 6> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<276>, ZPZV<194>, ZPZV<512>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type = DOLYVZPZV<1 | ZPZV<1 | Z
05164
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<851»; };
              template<> struct ConwayPolynomial<853, 8> { using ZPZ = aerobus::zpz<853>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<544>, ZPZV<846>, ZPZV<118>, ZPZV<118>, ZPZV<2»; }; //
        template<> struct ConwayPolynomial<853, 9> { using ZPZ = aerobus::zpz<853>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<851»;</pre>
05166
```

```
}; // NOLINT
05167
                template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
         POLYY<ZPZV<1>, ZPZV<854»; }; // NOLINT template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
05168
         POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
05169
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; }; // NOLINT
                template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
05170
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<528>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<857, 5> { using ZPZ = aerobus::zpz<857>; using type =
05171
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3$, ZPZV<3$, ZPZV<3$; }; // NOLINT template<> struct ConwayPolynomial<857, 6> { using ZPZ = aerobus::zpz<857>; using type =
05172
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<82>, ZPZV<65>, ZPZV<6s>, ZPZV<6s>, ZPZV<85>, ZPZV<85 , ZPZV
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<854»; }; // NOLINT
05174
               template<> struct ConwayPolynomial<857, 8> { using ZPZ = aerobus::zpz<857>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<611>, ZPZV<552>, ZPZV<494>, ZPZV<3»; }; //
         NOLINT
               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
05176
               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 =
05177
         POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT
                template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type =
05179
         template<> struct ConwayPolynomial<859, 5> { using ZPZ = aerobus::zpz<859>; using type =
05180
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<857»; }; // NOLINT
0.5181
                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<566>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpz<859>; using type =
05182
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 8> { using ZPZ = aerobus::zpz<859>; using type =
05183
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<522>, ZPZV<446>, ZPZV<672>, ZPZV<62»; }; //
05184
               template<> struct ConwayPolynomial<859, 9> { using ZPZ = aerobus::zpz<859>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<648>, ZPZV<845>, ZPZV<857»;
         }; // NOLINT
05185
               template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
         POLYV<ZPZV<1>, ZPZV<858»; }; // NOLINT
               template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
05186
         POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; }; // NOLINT
05187
               template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<858»; }; // NOLINT
template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<863, 5> { using ZPZ = aerobus::zpz<863>; using type =
05188
05189
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<858»; }; // NOLINT
               template<> struct ConwayPolynomial<863, 6> { using ZPZ = aerobus::zpz<863>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<330>, ZPZV<62>, ZPZV<300>, ZPZV<5»; }; // NOLINT
         template<> struct ConwayPolynomial<863, 7> { using ZPZ = aerobus::zpz<863>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<858»; }; // NOLINT
05191
               template<> struct ConwayPolynomial<863, 8> { using ZPZ = aerobus::zpz<863>; using type =
05192
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<765>, ZPZV<576>, ZPZV<849>, ZPZV<5»; }; //
         template<> struct ConwayPolynomial<863, 9> { using ZPZ = aerobus::zpz<863>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<381>, ZPZV<1>, ZPZV<858»; };
         // NOLINT
               template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
05194
         POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
         POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
05196
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875»; }; // NOLINT
template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; }; // NOLINT
05197
                template<> struct ConwayPolynomial<877, 5> { using ZPZ = aerobus::zpz<877>; using type =
05198
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<875»; }; // NOLINT
05199
               template<> struct ConwayPolynomial<877, 6> { using ZPZ = aerobus::zpz<877>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<400>, ZPZV<85>>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<877, 7> { using ZPZ = aerobus::zpz<877>; using type :
05200
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<675»; }; // NOLIN template<> struct ConwayPolynomial<877, 8> { using ZPZ = aerobus::zpz<877>; using type =
                                                                                                                                             // NOLINT
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<767>, ZPZV<319>, ZPZV<347>, ZPZV<2»; };
         template<> struct ConwayPolynomial<877, 9> { using ZPZ = aerobus::zpz<877>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<770>, ZPZV<278>, ZPZV<875»;</pre>
05202
         }; // NOLINT
                template<> struct ConwayPolynomial<881, 1> { using ZPZ = aerobus::zpz<881>; using type =
         POLYV<ZPZV<1>, ZPZV<878»; }; // NOLINT
05204
               template<> struct ConwayPolynomial<881, 2> { using ZPZ = aerobus::zpz<881>; using type =
         POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT
  template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
05205
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<878»; }; // NOLINT
```

```
05206
               template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<447>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<881, 5> { using ZPZ = aerobus::zpz<881>; using type =
05207
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<878»; }; // NOLINT
        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 =
05208
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
05210
              template<> struct ConwayPolynomial<881, 8> { using ZPZ = aerobus::zpz<881>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<635>, ZPZV<490>, ZPZV<561>, ZPZV<561>, ZPZV<3»; }; //
         NOLINT
              template<> struct ConwayPolynomial<881, 9> { using ZPZ = aerobus::zpz<881>; using type =
05211
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<587>, ZPZV<510>, ZPZV<878»;
         }; // NOLINT
05212
              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 =
05213
        POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<883, 3> { using ZPZ = aerobus::zpz<883>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<881»; }; // NOLINT
               template<> struct ConwayPolynomial<883, 4> { using ZPZ = aerobus::zpz<883>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<883, 5> { using ZPZ = aerobus::zpz<883>; using type =
0.5216
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<881»; }; // NOLINT
05217
               template<> struct ConwayPolynomial<883, 6> { using ZPZ = aerobus::zpz<883>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<879>, ZPZV<865>, ZPZV<871>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type =
05218
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<881»; }; // NOLINT template<> struct ConwayPolynomial<883, 8> { using ZPZ = aerobus::zpz<883>; using type =
05219
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<740>, ZPZV<762>, ZPZV<768>, ZPZV<2»; }; //
         NOLINT
05220
               template<> struct ConwayPolynomial<883, 9> { using ZPZ = aerobus::zpz<883>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<360>, ZPZV<557>, ZPZV<881»;
         }; // NOLINT
05221
              template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
        POLYV<ZPZV<1>, ZPZV<882»; }; // NOLINT
              template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
05222
        POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; }; // NOLINT
05223
               template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<882*; }; // NOLINT template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
05224
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<887, 5> { using ZPZ = aerobus::zpz<887>; using type =
05225
        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
05227
              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<882»; }; // NOLINT
05228
              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<5»; }; //
              template<> struct ConwayPolynomial<887, 9> { using ZPZ = aerobus::zpz<887>; using type =
05229
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<727>, ZPZV<345>, ZPZV<882»;
         }; // NOLINT
05230
              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 =
        POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<907, 3> { using ZPZ = aerobus::zpz<907>; using type =
05232
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<905»; }; // NOLINT template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
05233
        POLYVCZPZV<1>, ZPZV<1>, ZPZV<14>, ZPZV<478>, ZPZV<278; }; // NOLINT template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
05234
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; }; // NOLINT
05235
              template<> struct ConwayPolynomial<907, 6> { using ZPZ = aerobus::zpz<907>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<626>, ZPZV<752>, ZPZV<266>, ZPZV<2%; }; // NOLINT template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type
05236
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; }; // NOLINT
              template<> struct ConwayPolynomial<907, 8> { using ZPZ = aerobus::zpz<907>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<584>, ZPZV<518>, ZPZV<811>, ZPZV<2»; }; //
05238
             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<1>, ZPZV<783>, ZPZV<57>, ZPZV<905»;
         }; // NOLINT
05239
               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 =
05240
         POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; }; // NOLINT
              template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
05241
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; }; // NOLINT
              template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
05242
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17»; }; // NOLINT
               template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<894»; }; // NOLINT
05244
              template<> struct ConwayPolynomial<911, 6> { using ZPZ = aerobus::zpz<911>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<183>, ZPZV<19>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
05245
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<894»; };
                   template<> struct ConwayPolynomial<911, 8> { using ZPZ = aerobus::zpz<911>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<590>, ZPZV<168>, ZPZV<17»; }; //
           NOLINT
05247
                  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<67>, ZPZV<679>, ZPZV<894»;
           }; // NOLINT
05248
                   template<> struct ConwayPolynomial<919, 1> { using ZPZ = aerobus::zpz<919>; using type =
           POLYV<ZPZV<1>, ZPZV<912»; }; // NOLINT
                  template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
05249
           POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; }; // NOLINT
                  template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
05250
           POLYY<ZPZY<1>, ZPZV<0>, ZPZV<2>, ZPZV<912*; }; // NOLINT template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7»; }; // NOLINT
          template<> struct ConwayPolynomial<919, 5> { using ZPZ = aerobus::zpz<919>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<912»; }; // NOLINT</pre>
05252
           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
05253
                  template<> struct ConwayPolynomial<919,
                                                                                            7> { using ZPZ = aerobus::zpz<919>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9
05255
                  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<7»; }; //
           NOLINT
05256
                  template<> struct ConwayPolynomial<919, 9> { using ZPZ = aerobus::zpz<919>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<410>, ZPZV<623>, ZPZV<912»;
           }; // NOLINT
05257
                  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 =
05258
           POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3»; }; // NOLINT
05259
                   template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZV<926»; }; // NOLINT template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type =
05260
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
05261
           POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>; // NOLINT template<> struct ConwayPolynomial<929, 6> { using ZPZ = aerobus::zpz<929>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<805>, ZPZV<92>, ZPZV<86>, ZPZV<3»; };
                  template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type =
05263
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<926»; }; // NOLINT
                  template<> struct ConwayPolynomial<929, 8> { using ZPZ = aerobus::zpz<929>; using type =
05264
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699>, ZPZV<292>, ZPZV<586>, ZPZV<58); //
                  template<> struct ConwayPolynomial<929, 9> { using ZPZ = aerobus::zpz<929>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<481>, ZPZV<199>, ZPZV<926»;
           }; // NOLINT
05266
                  template<> struct ConwayPolynomial<937, 1> { using ZPZ = aerobus::zpz<937>; using type =
           POLYV<ZPZV<1>, ZPZV<932»; }; // NOLINT
                  template<> struct ConwayPolynomial<937, 2> { using ZPZ = aerobus::zpz<937>; using type =
05267
           POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5»; }; // 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 =
05269
          POLYV<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<5*; }; // NOLINT template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
05270
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<932»; }; // NOLINT
                  template<> struct ConwayPolynomial<937, 6> { using ZPZ = aerobus::zpz<937>; using type =
05271
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<794>, ZPZV<794>, ZPZV<934>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<937, 7> { using ZPZ = aerobus::zpz<937>; using type =
05272
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<23, ZPZV<24, ZPZV<32»; }; // NOLINT template<> struct ConwayPolynomial<937, 8> { using ZPZ = aerobus::zpz<937>; using type =
05273
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<265, ZPZV<53>, ZPZV<5»; };
05274
                 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
                  \texttt{template<> struct ConwayPolynomial<941, 1> \{ using ZPZ = aerobus:: zpz<941>; 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 |
05275
           POLYV<ZPZV<1>, ZPZV<939»; }; // NOLINT
05276
                   template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
           POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2»; }; // NOLINT
05277
                  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 =
05278
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2%; }; // NOLINT template<> struct ConwayPolynomial<941, 5> { using ZPZ = aerobus::zpz<941>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<939»; }; // NOLINT
05280
                   template<> struct ConwayPolynomial<941, 6> { using ZPZ = aerobus::zpz<941>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<459>, ZPZV<694>, ZPZV<538>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<941, 7> { using ZPZ = aerobus::zpz<941>; using type =
05281
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<939»; }; // NOLINT
                  template<> struct ConwayPolynomial<941, 8> { using ZPZ = aerobus::zpz<941>; using type =
05282
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<675>, ZPZV<590>, ZPZV<2»; };
           NOLINT
05283
                  template<> struct ConwayPolynomial<941, 9> { using ZPZ = aerobus::zpz<941>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708>, ZPZV<197>, ZPZV<939%;
           }; // NOLINT
```

```
05284
                  template<> struct ConwayPolynomial<947, 1> { using ZPZ = aerobus::zpz<947>; using type =
          POLYV<ZPZV<1>, ZPZV<945»; }; // NOLINT
05285
                 template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
          POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
05286
          POLYY<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
                 template<> struct ConwayPolynomial<947, 5> { using ZPZ = aerobus::zpz<947>; using type =
05288
          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 =
05289
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<880>, ZPZV<787>, ZPZV<95>, ZPZV<2»; }; // NOLINT
05290
                 template<> struct ConwayPolynomial<947,
                                                                                        7> { using ZPZ = aerobus::zpz<947>; using type
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<945»; }; //
05291
                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»; }; //
          NOT.TNT
          template<> struct ConwayPolynomial<947, 9> { using ZPZ = aerobus::zpz<947>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<30>, ZPZV<30
05292
                 template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<950»; }; // NOLINT
                 template<> struct ConwayPolynomial<953, 2> { using ZPZ = aerobus::zpz<953>; using type =
05294
          POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
05295
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950»; }; // NOLINT template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
05296
          05297
                  template<> struct ConwayPolynomial<953, 5> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV-ZPZV-1>, ZPZV-(>, ZPZV-(>, ZPZV-(>), ZPZV-(>), ZPZV-1>, ZPZV-(>50»; }; // NOLINT template<> struct ConwayPolynomial<953, 6> { using ZPZ = aerobus::zpz<953>; using type =
05298
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<507>, ZPZV<829>, ZPZV<730>, ZPZV<3»; };
                                                                                                                                                       // NOLINT
                 template<> struct ConwayPolynomial<953, 7> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
05300
                 template<> struct ConwayPolynomial<953, 8> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<579>, ZPZV<658>, ZPZV<108>, ZPZV<3»; }; //
          NOLINT
                 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
05302
                 template<> struct ConwayPolynomial<967, 1> { using ZPZ = aerobus::zpz<967>; using type =
          POLYV<ZPZV<1>, ZPZV<962»; }; // NOLINT
                 template<> struct ConwayPolynomial<967, 2> { using ZPZ = aerobus::zpz<967>; using type =
05303
          POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5»; }; // NOLINT
                  template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962»; };
                                                                                               // NOLINT
                template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
05305
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<967, 5> { using ZPZ = aerobus::zpz<967>; using type =
05306
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<962»; }; // NOLINT
05307
                  template<> struct ConwayPolynomial<967, 6> { using ZPZ = aerobus::zpz<967>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<948>, ZPZV<831>, ZPZV<5»; }; // NOLINT
05308
                 template<> struct ConwayPolynomial<967, 7> { using ZPZ = aerobus::zpz<967>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 8> { using ZPZ = aerobus::zpz<967>; using type =
05309
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<840>, ZPZV<502>, ZPZV<136>, ZPZV<5»; }; //
05310
                template<> struct ConwayPolynomial<967, 9> { using ZPZ = aerobus::zpz<967>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<512>, ZPZV<783>, ZPZV<783>, ZPZV<962»;
          }; // NOLINT
05311
                 template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<965»; }; // NOLINT
05312
                  template<> struct ConwayPolynomial<971, 2> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<970>, ZPZV<6»; }; // NOLINT
                 template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
05313
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<965»; }; // NOLINT
template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<52>, ZPZV<6»; }; // NOLINT
template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
05314
05315
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<965»; }; // NOLINT
                 template<> struct ConwayPolynomial<971, 6> { using ZPZ = aerobus::zpz<971>; using type =
05316
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<970>, ZPZV<729>, ZPZV<718>, ZPZV<6»; }; // NOLINT
05317
                 template<> struct ConwayPolynomial<971, 7> { using ZPZ = aerobus::zpz<971>; using type
          POLYY<ZPZV<1>, ZPZV<0>, 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 =
05318
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<725>, ZPZV<281>, ZPZV<206>, ZPZV<6»; }; //
05319
                 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
05320
                  template<> struct ConwayPolynomial<977, 1> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<974»; };
                                                               // NOLINT
                  template<> struct ConwayPolynomial<977, 2> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; }; // NOLINT
                template<> struct ConwayPolynomial<977, 3> { using ZPZ = aerobus::zpz<977>; using type =
05322
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<974»; }; // NOLINT template<> struct ConwayPolynomial<977, 4> { using ZPZ = aerobus::zpz<977>; using type =
05323
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<800>, ZPZV<3»; };
            template<> struct ConwayPolynomial<977, 5> { using ZPZ = aerobus::zpz<977>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<974»; }; // NOLINT
           template<> struct ConwayPolynomial<977, 6> { using ZPZ = aerobus::zpz<977>; using type =
05325
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<729>, ZPZV<830>, ZPZV<753>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<977,
                                                           7> { using ZPZ = aerobus::zpz<977>; using type
05326
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<974»; }; //
05327
           template<> struct ConwayPolynomial<977, 8> { using ZPZ = aerobus::zpz<977>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<855>, ZPZV<807>, ZPZV<77>, ZPZV<3»; };
05328
           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<450>, ZPZV<740>, ZPZV<974»;
       }; // NOLINT
   template<>> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
       POLYV<ZPZV<1>, ZPZV<978»; }; // NOLINT
05330
           template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
05331
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978»; }; // NOLINT
           template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5»; }; // NOLINT
05333
           template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<978»; }; // NOLINT template<> struct ConwayPolynomial<983, 6> { using ZPZ = aerobus::zpz<983>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<849>, ZPZV<296>, ZPZV<228>, ZPZV<5»; }; // NOLINT
05334
            template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<978»; };
           template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<5»; }; //
       NOLINT
           template<> struct ConwayPolynomial<983, 9> { using ZPZ = aerobus::zpz<983>; using type =
05337
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<87>, ZPZV<978»;
       }; // NOLINT
           template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
05338
       POLYV<ZPZV<1>, ZPZV<985»; }; // NOLINT
           template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
05339
       POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; }; // NOLINT
            template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985»; }; // NOLINT template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
05341
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6»; }; // NOLINT
template<> struct ConwayPolynomial<991, 5> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<985»; }; // NOLINT
05342
05343
           template<> struct ConwayPolynomial<991, 6> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<637>, ZPZV<278>, ZPZV<278>, ZPZV<278>, ZPZV<291>; // NOLINT template<> struct ConwayPolynomial<991, 7> { using ZPZ = aerobus::zpz<991>; using type =
05344
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<985»; }; // NOLINT template<> struct ConwayPolynomial<991, 8> { using ZPZ = aerobus::zpz<991>; using type =
05345
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<941>, ZPZV<786>, ZPZV<234>, ZPZV<6»; }; //
       NOLINT
05346
           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<9>, ZPZV<466>, ZPZV<222>, ZPZV<985»;
       }; // NOLINT
05347
           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 =
05348
       POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<990»; }; // NOLINT template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
05350
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<62>, ZPZV<622>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
05351
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<990»; }; // NOLINT
            template<> struct ConwayPolynomial<997, 6> { using ZPZ = aerobus::zpz<997>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<981>, ZPZV<58>, ZPZV<260>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<997, 7> { using ZPZ = aerobus::zpz<997>; using type =
05353
      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 =
05354
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<473>, ZPZV<241>, ZPZV<241>, ZPZV<7»; }; //
       NOLINT
           template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type =
05355
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<39>, ZPZV<732>, ZPZV<616>, ZPZV<990»;
       }; // NOLINT
05356 #endif // DO NOT DOCUMENT
         // namespace aerobus
05357 }
05358 #endif // AEROBUS_CONWAY_IMPORTS
05359
05360 #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>

192 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 193

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

# 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 < 6>, i32::val < 15>>

v1	a value in i32
v2	a value in i32

194 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 195

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

### 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>

### 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>	

196 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<1>

### 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<2>	

10.27 i64::pos\_v 197

#### 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>>

198 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 Pl\_fraction::val

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

### 10.32 E\_fraction::val

approximation of e -> 2.718...

approximation of  $e \rightarrow 2.718...$ 

## Index

```
abs t
                                                              pq64, 30
                                                              q32, 30
     aerobus, 21
add t
                                                              q64, 30
    aerobus::i32, 58
                                                              sin, 30
    aerobus::i64, 62
                                                              sinh, 31
    aerobus::polynomial < Ring >, 67
                                                              SQRT2_fraction, 31
    aerobus::Quotient < Ring, X >, 74
                                                              SQRT3 fraction, 31
    aerobus::zpz, 98
                                                              stirling_signed_t, 31
addfractions_t
                                                              stirling_unsigned_t, 31
                                                              tan, 32
    aerobus, 21
aerobus, 17
                                                              tanh, 32
    abs t, 21
                                                              taylor, 32
    addfractions_t, 21
                                                              vadd_t, 32
    aligned_malloc, 33
                                                              vmul t, 33
    alternate t, 22
                                                         aerobus::ContinuedFraction < a0 >, 49
    alternate_v, 34
                                                              type, 49
                                                              val, 49
    asin, 22
    asinh, 22
                                                         aerobus::ContinuedFraction < a0, rest... >, 50
    atan, 23
                                                              type, 50
    atanh, 23
                                                              val, 51
                                                         aerobus::ContinuedFraction < values >, 48
    bell t, 23
    bernoulli t, 23
                                                         aerobus::ConwayPolynomial, 51
    bernoulli v, 34
                                                         aerobus::Embed< i32, i64 >, 52
    combination t, 24
                                                              type, 52
                                                         aerobus::Embed< polynomial< Small >, polynomial<
    combination_v, 34
    cos, 24
                                                                  Large > >, 52
    cosh, 24
                                                              type, 53
    E_fraction, 24
                                                         aerobus::Embed< q32, q64 >, 53
    embed_int_poly_in_fractions_t, 24
                                                              type, 54
                                                         aerobus::Embed< Quotient< Ring, X >, Ring >, 54
    exp, 26
    expm1, 26
                                                         aerobus::Embed< Ring, FractionField< Ring >>, 55
    factorial_t, 26
    factorial_v, 34
                                                              type, 56
    field, 33
                                                         aerobus::Embed< Small, Large, E >, 51
    fpq32, 26
                                                         aerobus::Embed< zpz< x>, i32>, 56
    fpq64, 27
                                                              type, 57
    FractionField, 27
                                                         aerobus::i32, 57
                                                              add t, 58
    gcd t, 27
    geometric_sum, 27
                                                              div_t, 58
    Inp1, 27
                                                              eq_t, 58
    make_frac_polynomial_t, 28
                                                              eq_v, 60
    make int polynomial t, 28
                                                              gcd t, 58
    make q32 t, 28
                                                              gt t, 59
    make_q64_t, 28
                                                              inject constant t, 59
    makefraction t, 29
                                                              inject ring t, 59
    mulfractions t, 29
                                                              inner type, 59
    pi64, 29
                                                              is_euclidean_domain, 60
    PI_fraction, 29
                                                              is field, 60
    pow_t, 30
                                                              It_t, 59
```

mod_t, 59	chebyshev_U, 41
mul_t, 59	hermite_kind, 43
one, 60	hermite_phys, 41
pos_t, 60	hermite_prob, 41
pos_v, 60	laguerre, 42
sub_t, 60	legendre, 42
zero, 60	physicist, 43
aerobus::i32::val< x >, 82	probabilist, 43
enclosing_type, 83	aerobus::polynomial < Ring >, 66
eval, 84	add_t, 67
get, 84	derive_t, 68
is_zero_t, 83	div_t, 68
to_string, 84	eq_t, 68
v, 84	gcd_t, 68
aerobus::i64, 61	gt_t, 69
add_t, 62	inject_constant_t, 69
div_t, 62	inject_ring_t, 69
eq_t, 62	is_euclidean_domain, 72
eq_v, 64	is field, 72
gcd_t, 62	lt_t, 69
gt_t, 62	mod_t, 69
gt_v, 64	monomial t, 70
inject_constant_t, 62	mul_t, 70
inject_ring_t, 63	one, 70
inner_type, 63	pos_t, 70
is_euclidean_domain, 64	pos_v, 72
is_field, 64	simplify_t, 71
It_t, 63	sub_t, 71
It_v, 64	X, 71
mod_t, 63	zero, 71
mul_t, 63	aerobus::polynomial< Ring >::val< coeffN >, 94
one, 63	aN, 95
pos_t, 63	coeff_at_t, 95
pos_v, 65	degree, 96
sub_t, 64	enclosing_type, 95
zero, 64	eval, 96
aerobus::i64::val< x >, 85	is_zero_t, 95
enclosing_type, 86	is_zero_v, 96
eval, 86	ring_type, 95
get, 86	strip, 95
inner_type, 86	to string, 96
is_zero_t, 86	aerobus::polynomial< Ring >::val< coeffN >::coeff_at<
to_string, 87	index, $E > 47$
v, 87	aerobus::polynomial< Ring >::val< coeffN >::coeff_at<
aerobus::internal, 35	index, std::enable_if_t<(index< 0     index >
index_sequence_reverse, 38	0)>>, 47
is_instantiation_of_v, 39	type, 47
make_index_sequence_reverse, 38	aerobus::polynomial < Ring >::val < coeffN >::coeff_at <
type_at_t, 38	index, std::enable_if_t<(index==0)>>, 48
aerobus::is_prime $<$ n $>$ , 65	type, 48
value, 65	aerobus::polynomial< Ring >::val< coeffN, coeffs >,
aerobus::IsEuclideanDomain, 45	87
aerobus::IsField, 45	aN, 88
aerobus::IsRing, 46	coeff_at_t, 88
aerobus::known_polynomials, 39	degree, 90
bernoulli, 40	enclosing_type, 89
bernotein, 40	eval, 89
chebyshev_T, 40	is_zero_t, 89
51105 y 5110 v_1, TO	10_2010_1, 00

is_zero_v, 90	pos_t, 101
ring_type, 89	pos_v, 103
strip, 89	sub_t, 101
to_string, 90	zero, 101
aerobus::Quotient $<$ Ring, $X>$ , 73	aerobus:: $zpz $ :: $val < x >$ , 91
add_t, 74	enclosing_type, 92
div_t, 75	eval, 93
eq_t, 75	get, 93
eq_v, 77	is_zero_t, 92
inject_constant_t, 75	is_zero_v, 93
inject_ring_t, 75	to_string, 93
is_euclidean_domain, 77	v, 93
mod_t, 75	aligned_malloc
mul_t, 76	aerobus, 33
one, 76	alternate_t
pos_t, 76	aerobus, 22
pos_v, 77	alternate_v
zero, 76	aerobus, 34
aerobus::Quotient< Ring, X >::val< V >, 91	aN
raw_t, 91	aerobus::polynomial< Ring >::val< coeffN >, 95
type, 91	aerobus::polynomial< Ring >::val< coeffN, coeffs
aerobus::type_list< Ts >, 78	>, 88
at, 79	asin
concat, 80	aerobus, 22
insert, 80	asinh
length, 81	aerobus, 22
push_back, 80	at
push_front, 80	aerobus::type_list< Ts >, 79
remove, 81	atan
aerobus::type_list< Ts >::pop_front, 72	aerobus, 23
tail, 73	atanh
type, 73	aerobus, 23
aerobus::type_list< Ts >::split< index >, 78	
head, 78	bell_t
tail, 78	aerobus, 23
aerobus::type_list<>, 81	bernoulli
concat, 82	aerobus::known_polynomials, 40
insert, 82	bernoulli_t
length, 82	aerobus, 23
push_back, 82	bernoulli_v
push_front, 82	aerobus, 34
aerobus::zpz, 96	bernstein
add_t, 98	aerobus::known_polynomials, 40
div_t, 98	chebyshev_T
eq_t, 99	aerobus::known polynomials, 40
eq_v, 102	chebyshev_U
gcd_t, 99	aerobus::known_polynomials, 41
gt_t, 99	coeff_at_t
gt_v, 102	aerobus::polynomial< Ring >::val< coeffN >, 95
inject_constant_t, 99	aerobus::polynomial < Ring >::val < coeffN, coeffs
inner_type, 100	>, 88
is_euclidean_domain, 102	combination_t
is_field, 102	aerobus, 24
lt_t, 100	combination_v
lt_v, 102	aerobus, 34
mod_t, 100	concat
mul_t, 100	aerobus::type_list< Ts >, 80
one, 101	aerobus::type_list<>, 82

cos	aerobus, 26
aerobus, 24	fpq64
cosh	aerobus, 27
aerobus, 24	FractionField
degree	aerobus, 27
aerobus::polynomial< Ring >::val< coeffN >, 96	gcd_t
aerobus::polynomial < Ring >::val < coeffN, coeffs	aerobus, 27
>, 90	aerobus::i32, 58
derive_t	aerobus::i64, 62
aerobus::polynomial < Ring >, 68	aerobus::polynomial< Ring >, 68
div_t	aerobus::zpz, 99
aerobus::i32, 58	geometric_sum
aerobus::i64, 62	aerobus, 27
aerobus::polynomial < Ring >, 68	get
aerobus::Quotient< Ring, X >, 75	aerobus::i32::val $<$ x $>$ , 84
aerobus::zpz, 98	aerobus::i64::val< x >, 86
E fraction	aerobus:: $zpz ::val < x >, 93$
aerobus, 24	gt_t
embed_int_poly_in_fractions_t	aerobus::i32, 59 aerobus::i64, 62
aerobus, 24	aerobus::polynomial < Ring >, 69
enclosing_type	aerobus::zpz, 99
aerobus::i32::val < $x >$ , 83	gt_v
aerobus::i64::val< x >, 86	aerobus::i64, 64
aerobus::polynomial < Ring >::val < coeffN >, 95	aerobus::zpz $<$ p $>$ , 102
aerobus::polynomial< Ring >::val< coeffN, coeffs	, ,
>, 89	head
aerobus:: $zpz ::val < x >$ , 92	aerobus::type_list< Ts >::split< index >, 78
ea t	hermite_kind
eq_t	
aerobus::i32, 58	aerobus::known_polynomials, 43
aerobus::i32, 58 aerobus::i64, 62	aerobus::known_polynomials, 43 hermite_phys
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68	aerobus::known_polynomials, 43 hermite_phys aerobus::known_polynomials, 41
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75	aerobus::known_polynomials, 43 hermite_phys aerobus::known_polynomials, 41 hermite_prob
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99	aerobus::known_polynomials, 43 hermite_phys aerobus::known_polynomials, 41
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99 eq_v	aerobus::known_polynomials, 43 hermite_phys aerobus::known_polynomials, 41 hermite_prob aerobus::known_polynomials, 41
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99 eq_v aerobus::i32, 60	aerobus::known_polynomials, 43 hermite_phys aerobus::known_polynomials, 41 hermite_prob
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99 eq_v aerobus::i32, 60 aerobus::i64, 64	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41 index_sequence_reverse
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99 eq_v aerobus::i32, 60 aerobus::i64, 64 aerobus::Quotient < Ring, X >, 77	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41 index_sequence_reverse     aerobus::internal, 38
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99 eq_v aerobus::i32, 60 aerobus::i64, 64	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99  eq_v aerobus::i32, 60 aerobus::i64, 64 aerobus::Quotient < Ring, X >, 77 aerobus::zpz , 102	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::polynomial < Ring >, 69
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99  eq_v aerobus::i32, 60 aerobus::i64, 64 aerobus::Quotient < Ring, X >, 77 aerobus::zpz , 102 eval	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::polynomial < Ring >, 69     aerobus::Quotient < Ring, X >, 75
aerobus::i32, 58     aerobus::i64, 62     aerobus::polynomial < Ring >, 68     aerobus::Quotient < Ring, X >, 75     aerobus::zpz , 99  eq_v     aerobus::i32, 60     aerobus::i64, 64     aerobus::Quotient < Ring, X >, 77     aerobus::zpz , 102  eval     aerobus::i32::val < $x >$ , 84     aerobus::i64::val < $x >$ , 86     aerobus::polynomial < Ring >::val < coeffN >, 96	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::polynomial< Ring >, 69     aerobus::Quotient< Ring, X >, 75     aerobus::zpz, 99
aerobus::i32, 58     aerobus::i64, 62     aerobus::polynomial $<$ Ring $>$ , 68     aerobus::Quotient $<$ Ring, $X >$ , 75     aerobus::zpz $<$ p $>$ , 99  eq_v     aerobus::i32, 60     aerobus::i64, 64     aerobus::Quotient $<$ Ring, $X >$ , 77     aerobus::zpz $<$ p $>$ , 102  eval  aerobus::i32::val $<$ x $>$ , 84     aerobus::i64::val $<$ x $>$ , 86     aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN $>$ , 96     aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN, coeffs	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::polynomial< Ring >, 69     aerobus::Quotient< Ring, X >, 75     aerobus::zpz, 99 inject_ring_t
aerobus::i32, 58     aerobus::i64, 62     aerobus::polynomial $<$ Ring $>$ , 68     aerobus::Quotient $<$ Ring, $X >$ , 75     aerobus::zpz $<$ p $>$ , 99  eq_v     aerobus::i32, 60     aerobus::i64, 64     aerobus::Quotient $<$ Ring, $X >$ , 77     aerobus::zpz $<$ p $>$ , 102  eval  aerobus::i32::val $<$ x $>$ , 84     aerobus::i64::val $<$ x $>$ , 86     aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN $>$ , 96     aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN, coeffs $>$ , 89	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::polynomial < Ring >, 69     aerobus::Quotient < Ring, X >, 75     aerobus::zpz , 99 inject_ring_t     aerobus::i32, 59
aerobus::i32, 58     aerobus::i64, 62     aerobus::polynomial $<$ Ring $>$ , 68     aerobus::Quotient $<$ Ring, $X >$ , 75     aerobus:: $zpz $ , 99  eq_v  aerobus::i32, 60     aerobus::i44, 64     aerobus::Quotient $<$ Ring, $x >$ , 77     aerobus:: $zpz $ , 102  eval  aerobus::i32::val $<$ $x >$ , 84     aerobus::i64::val $<$ $x >$ , 86     aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN $>$ , 96     aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN, coeffs $>$ , 89     aerobus:: $zpz $ ::val $<$ $x >$ , 93	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::polynomial< Ring >, 69     aerobus::Quotient< Ring, X >, 75     aerobus::zpz, 99 inject_ring_t     aerobus::i32, 59     aerobus::i34, 63
aerobus::i32, 58     aerobus::i64, 62     aerobus::polynomial $<$ Ring $>$ , 68     aerobus::Quotient $<$ Ring, $X >$ , 75     aerobus::Zpz $<$ p $>$ , 99  eq_v  aerobus::i32, 60     aerobus::i64, 64     aerobus::Quotient $<$ Ring, $X >$ , 77     aerobus::zpz $<$ p $>$ , 102  eval  aerobus::i32::val $<$ x $>$ , 84     aerobus::i64::val $<$ x $>$ , 86     aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN $>$ , 96     aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN, coeffs $>$ , 89     aerobus::zpz $<$ p $>$ ::val $<$ x $>$ , 93  exp	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::polynomial< Ring >, 69     aerobus::Quotient< Ring, X >, 75     aerobus::zpz, 99 inject_ring_t     aerobus::i32, 59     aerobus::i64, 63     aerobus::polynomial< Ring >, 69
aerobus::i32, 58     aerobus::i64, 62     aerobus::polynomial < Ring >, 68     aerobus::Quotient < Ring, X >, 75     aerobus::zpz , 99  eq_v     aerobus::i32, 60     aerobus::i64, 64     aerobus::Quotient < Ring, X >, 77     aerobus::zpz , 102  eval     aerobus::i32::val < x >, 84     aerobus::i64::val < x >, 86     aerobus::polynomial < Ring >::val < coeffN >, 96     aerobus::polynomial < Ring >::val < coeffN, coeffs >, 89     aerobus::zpz ::val < x >, 93  exp     aerobus, 26	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::polynomial< Ring >, 69     aerobus::quotient< Ring, X >, 75     aerobus::zpz, 99 inject_ring_t     aerobus::i32, 59     aerobus::i64, 63     aerobus::polynomial< Ring >, 69     aerobus::i94, 63     aerobus::polynomial< Ring >, 69     aerobus::quotient< Ring, X >, 75
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99  eq_v aerobus::i32, 60 aerobus::i64, 64 aerobus::Quotient < Ring, X >, 77 aerobus::zpz , 102  eval aerobus::i32::val < x >, 84 aerobus::i64::val < x >, 86 aerobus::polynomial < Ring >::val < coeffN >, 96 aerobus::polynomial < Ring >::val < coeffN, coeffs >, 89 aerobus::zpz ::val < x >, 93  exp aerobus, 26 expm1	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::polynomial< Ring >, 69     aerobus::Quotient< Ring, X >, 75     aerobus::zpz, 99 inject_ring_t     aerobus::i64, 63     aerobus::polynomial< Ring >, 69     aerobus::i9lynomial< Ring >, 69     aerobus::i9lynomial< Ring >, 69     aerobus::i9lynomial< Ring >, 69     aerobus::Quotient< Ring, X >, 75 inner_type
aerobus::i32, 58     aerobus::i64, 62     aerobus::polynomial < Ring >, 68     aerobus::Quotient < Ring, X >, 75     aerobus::zpz , 99  eq_v     aerobus::i32, 60     aerobus::i64, 64     aerobus::Quotient < Ring, X >, 77     aerobus::zpz , 102  eval     aerobus::i32::val < x >, 84     aerobus::i64::val < x >, 86     aerobus::polynomial < Ring >::val < coeffN >, 96     aerobus::polynomial < Ring >::val < coeffN, coeffs >, 89     aerobus::zpz ::val < x >, 93  exp     aerobus, 26	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::polynomial< Ring >, 69     aerobus::Quotient< Ring, X >, 75     aerobus::zpz, 99 inject_ring_t     aerobus::polynomial< Ring >, 69     aerobus::i64, 63     aerobus::polynomial< Ring >, 69     aerobus::Quotient< Ring, X >, 75 inject_ring_t     aerobus::Quotient< Ring, X >, 75 inner_type     aerobus::i32, 59
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99  eq_v aerobus::i32, 60 aerobus::i64, 64 aerobus::Quotient < Ring, X >, 77 aerobus::zpz , 102  eval aerobus::i32::val < x >, 84 aerobus::i64::val < x >, 86 aerobus::polynomial < Ring >::val < coeffN >, 96 aerobus::polynomial < Ring >::val < coeffN, coeffs >, 89 aerobus::zpz ::val < x >, 93  exp aerobus, 26  expm1 aerobus, 26	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::polynomial< Ring >, 69     aerobus::Quotient< Ring, X >, 75     aerobus::zpz, 99 inject_ring_t     aerobus::i32, 59     aerobus::j64, 63     aerobus::polynomial< Ring >, 69     aerobus::i32, 59     aerobus::Quotient< Ring, X >, 75 inner_type     aerobus::i32, 59     aerobus::i32, 59     aerobus::i64, 63
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99  eq_v aerobus::i32, 60 aerobus::i64, 64 aerobus::Quotient < Ring, X >, 77 aerobus::zpz , 102  eval aerobus::i32::val < x >, 84 aerobus::i64::val < x >, 86 aerobus::polynomial < Ring >::val < coeffN >, 96 aerobus::polynomial < Ring >::val < coeffN, coeffs >, 89 aerobus::zpz ::val < x >, 93  exp aerobus, 26 expm1	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::polynomial< Ring >, 69     aerobus::Quotient< Ring, X >, 75     aerobus::zpz, 99 inject_ring_t     aerobus::polynomial< Ring >, 69     aerobus::i64, 63     aerobus::polynomial< Ring >, 69     aerobus::Quotient< Ring, X >, 75 inject_ring_t     aerobus::Quotient< Ring, X >, 75 inner_type     aerobus::i32, 59
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99  eq_v aerobus::i32, 60 aerobus::i64, 64 aerobus::Quotient < Ring, X >, 77 aerobus::zpz , 102  eval aerobus::i32::val < x >, 84 aerobus::polynomial < Ring >::val < coeffN >, 96 aerobus::polynomial < Ring >::val < coeffN, coeffs >, 89 aerobus::zpz ::val < x >, 93  exp aerobus, 26  factorial_t	aerobus::known_polynomials, 43 hermite_phys
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99  eq_v aerobus::i32, 60 aerobus::i64, 64 aerobus::Quotient < Ring, X >, 77 aerobus::zpz , 102  eval aerobus::i32::val < x >, 84 aerobus::i64::val < x >, 86 aerobus::polynomial < Ring >::val < coeffN >, 96 aerobus::polynomial < Ring >::val < coeffN, coeffs >, 89 aerobus::zpz ::val < x >, 93  exp aerobus, 26  factorial_t aerobus, 26	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::Quotient< Ring, X >, 75     aerobus::zpz, 99 inject_ring_t     aerobus::i32, 59     aerobus::i64, 63     aerobus::i04, 63     aerobus::Quotient< Ring, X >, 75 inner_type     aerobus::Quotient< Ring, X >, 75 inner_type     aerobus::i32, 59     aerobus::i64, 63
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99  eq_v aerobus::i32, 60 aerobus::i64, 64 aerobus::Quotient < Ring, X >, 77 aerobus::zpz , 102  eval aerobus::i32::val < x >, 84 aerobus::i64::val < x >, 86 aerobus::polynomial < Ring >::val < coeffN >, 96 aerobus::polynomial < Ring >::val < coeffN, coeffs >, 89 aerobus::zpz ::val < x >, 93  exp aerobus, 26  factorial_t aerobus, 26  factorial_v	aerobus::known_polynomials, 43 hermite_phys     aerobus::known_polynomials, 41 hermite_prob     aerobus::known_polynomials, 41  index_sequence_reverse     aerobus::internal, 38 inject_constant_t     aerobus::i32, 59     aerobus::i64, 62     aerobus::Quotient< Ring, X >, 75     aerobus::zpz, 99 inject_ring_t     aerobus::i32, 59     aerobus::i64, 63     aerobus::polynomial< Ring >, 69     aerobus::i64, 63     aerobus::i64, 63     aerobus::quotient< Ring, X >, 75 inner_type     aerobus::i32, 59     aerobus::i64, 63     aerobus::i64, 63
aerobus::i32, 58 aerobus::i64, 62 aerobus::polynomial < Ring >, 68 aerobus::Quotient < Ring, X >, 75 aerobus::zpz , 99  eq_v aerobus::i32, 60 aerobus::i64, 64 aerobus::Quotient < Ring, X >, 77 aerobus::zpz , 102  eval aerobus::i32::val < x >, 84 aerobus::i64::val < x >, 86 aerobus::polynomial < Ring >::val < coeffN >, 96 aerobus::polynomial < Ring >::val < coeffN, coeffs >, 89 aerobus::zpz ::val < x >, 93  exp aerobus, 26  factorial_t aerobus, 26  factorial_v aerobus, 34	aerobus::known_polynomials, 43 hermite_phys

	aerobus::i32, 60		aerobus::Quotient< Ring, X >, 75
	aerobus::i64, 64		aerobus::zpz, 100
	aerobus::polynomial < Ring >, 72	mor	nomial_t
	aerobus::Quotient< Ring, X >, 77		aerobus::polynomial< Ring >, 70
	aerobus::zpz, 102	mul	
is fi		-	 aerobus::i32, 59
_	aerobus::i32, 60		aerobus::i64, 63
	aerobus::i64, 64		aerobus::polynomial< Ring >, 70
	aerobus::polynomial< Ring >, 72		aerobus::Quotient< Ring, X >, 76
	aerobus::zpz, 102		aerobus:: $zpz $ , 100
is ir	nstantiation_of_v	mul	fractions_t
.0	aerobus::internal, 39		aerobus, 29
is 7	ero t		40,000,20
.0	aerobus::i32::val< x >, 83	one	
	aerobus::i64::val < x > , 86		aerobus::i32, 60
	aerobus::polynomial< Ring >::val< coeffN >, 95		aerobus::i64, 63
			aerobus::polynomial< Ring >, 70
	aerobus::polynomial < Ring >::val < coeffN, coeffs		aerobus::Quotient< Ring, X >, 76
	>, 89		aerobus:: $zpz $ , 101
	aerobus::zpz $<$ p $>$ ::val $<$ x $>$ , 92		αστοράσ2ρ2 < ρ >, το τ
IS_Z	ero_v	phy	sicist
	aerobus::polynomial < Ring >::val < coeffN >, 96	Je J	aerobus::known_polynomials, 43
	aerobus::polynomial< Ring >::val< coeffN, coeffs	pi64	_ ·
	>, 90	pio i	aerobus, 29
	aerobus::zpz $<$ p $>$ ::val $<$ x $>$ , 93	PI f	raction
		' '_'	aerobus, 29
lagu		noc	
	aerobus::known_polynomials, 42	pos	_· aerobus::i32, 60
lege	endre		
	aerobus::known_polynomials, 42		aerobus::i64, 63
leng			aerobus::polynomial < Ring >, 70
	aerobus::type_list< Ts >, 81		aerobus::Quotient< Ring, X >, 76
	aerobus::type_list<>, 82		aerobus::zpz, 101
Inp1		pos	
	aerobus, 27		aerobus::i32, 60
lt_t			aerobus::i64, 65
	aerobus::i32, 59		aerobus::polynomial < Ring >, 72
	aerobus::i64, 63		aerobus::Quotient< Ring, X >, 77
	aerobus::polynomial< Ring >, 69		aerobus:: $zpz $ , 103
	aerobus::zpz $<$ p $>$ , 100	pow	_
lt_v			aerobus, 30
	aerobus::i64, 64	pq6	4
	aerobus::zpz, 102		aerobus, 30
		prob	pabilist
mak	re_frac_polynomial_t		aerobus::known_polynomials, 43
	aerobus, 28	pus	h_back
mak	re_index_sequence_reverse		aerobus::type_list< Ts >, 80
	aerobus::internal, 38		aerobus::type_list<>, 82
mak	re_int_polynomial_t	pus	h_front
	aerobus, 28		aerobus::type_list< Ts >, 80
mak	re_q32_t		aerobus::type_list<>, 82
	aerobus, 28		
mak	xe_q64_t	q32	
	aerobus, 28		aerobus, 30
mak	refraction_t	q64	
	aerobus, 29		aerobus, 30
mod			
	aerobus::i32, 59	raw	_t
	aerobus::i64, 63		aerobus::Quotient $<$ Ring, X $>$ ::val $<$ V $>$ , 91
	aerobus::polynomial< Ring >, 69	REA	ADME.md, 105

```
remove
                                                             aerobus::Embed< zpz< x>, i32>, 57
                                                             aerobus::polynomial< Ring >::val<
     aerobus::type_list< Ts >, 81
                                                                 >::coeff_at< index, std::enable_if_t<(index<
ring_type
     aerobus::polynomial < Ring >::val < coeffN >, 95
                                                                 0 \mid | \text{index} > 0) > , 47
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                             aerobus::polynomial< Ring
                                                                                            >::val<
                                                                 >::coeff at< index, std::enable_if_t<(index==0)>
         >, 89
simplify t
                                                             aerobus::Quotient < Ring, X >::val < V >, 91
     aerobus::polynomial < Ring >, 71
                                                             aerobus::type list< Ts >::pop front, 73
sin
                                                        type at t
     aerobus, 30
                                                             aerobus::internal, 38
sinh
     aerobus, 31
SQRT2_fraction
                                                             aerobus::i32::val < x >, 84
     aerobus, 31
                                                             aerobus::i64::val < x >, 87
SQRT3 fraction
                                                             aerobus::zpz ::val < x >, 93
     aerobus, 31
                                                        vadd t
src/aerobus.h, 105
                                                             aerobus, 32
stirling_signed_t
                                                        val
     aerobus, 31
                                                             aerobus::ContinuedFraction < a0 >, 49
stirling_unsigned_t
                                                             aerobus::ContinuedFraction < a0, rest... >, 51
     aerobus, 31
                                                        value
strip
                                                             aerobus::is_prime< n >, 65
     aerobus::polynomial < Ring >::val < coeffN >, 95
                                                        vmul t
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                             aerobus, 33
         >, 89
                                                        Χ
sub t
                                                             aerobus::polynomial < Ring >, 71
     aerobus::i32, 60
     aerobus::i64, 64
                                                        zero
     aerobus::polynomial < Ring >, 71
                                                             aerobus::i32, 60
     aerobus::zpz, 101
                                                             aerobus::i64, 64
                                                             aerobus::polynomial < Ring >, 71
tail
                                                             aerobus::Quotient < Ring, X >, 76
     aerobus::type list< Ts >::pop front, 73
                                                             aerobus::zpz, 101
     aerobus::type list< Ts >::split< index >, 78
tan
     aerobus, 32
tanh
     aerobus, 32
taylor
     aerobus, 32
to string
     aerobus::i32::val< x >, 84
     aerobus::i64::val < x >, 87
     aerobus::polynomial < Ring >::val < coeffN >, 96
     aerobus::polynomial< Ring >::val< coeffN, coeffs
          >, 90
     aerobus::zpz ::val < x >, 93
     aerobus::ContinuedFraction< a0 >, 49
     aerobus::ContinuedFraction< a0, rest... >, 50
     aerobus::Embed< i32, i64 >, 52
     aerobus::Embed< polynomial<
                                          Small
         polynomial < Large > >, 53
     aerobus::Embed < q32, q64 >, 54
     aerobus::Embed< Quotient< Ring, X >, Ring >,
     aerobus::Embed< Ring, FractionField< Ring >>,
         56
```