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	2
	1.2.1 Predefined discrete euclidean domains	2
	1.2.2 Polynomials	3
	1.2.3 Known polynomials	3
	1.2.4 Conway polynomials	3
	1.2.5 Taylor series	4
	1.3 Operations	5
	1.3.1 Field of fractions	5
	1.3.2 Quotient	6
	1.4 Misc	6
	1.4.1 Continued Fractions	6
	1.5 CUDA	6
•	Manager Indian	_
2	Namespace Index	7
	2.1 Namespace List	7
3	Concept Index	9
	3.1 Concepts	9
4	Class Index	11
	4.1 Class List	11
5	File Index	13
	5.1 File List	13
6	Namespace Documentation	15
	6.1 aerobus Namespace Reference	15
	6.1.1 Detailed Description	19
	6.1.2 Typedef Documentation	20
	6.1.2.1 abs_t	20
	6.1.2.2 add_t	20
	6.1.2.3 addfractions_t	20
	6.1.2.4 alternate_t	20
	6.1.2.5 asin	21
	6.1.2.6 asinh	21
	6.1.2.7 atan	21
	6.1.2.8 atanh	21
	6.1.2.9 bell_t	23
	6.1.2.10 bernoulli_t	23
	6.1.2.11 combination_t	23

6.1.2.12 cos		23
6.1.2.13 cosh		25
6.1.2.14 div_t		25
6.1.2.15 E_fraction		25
6.1.2.16 embed_int_poly_in_fractions_t		25
6.1.2.17 exp		26
6.1.2.18 expm1		26
6.1.2.19 factorial_t		26
6.1.2.20 fpq32		26
6.1.2.21 fpq64		27
6.1.2.22 FractionField		27
6.1.2.23 gcd_t		27
6.1.2.24 geometric_sum		27
6.1.2.25 lnp1		28
6.1.2.26 make_frac_polynomial_t		28
6.1.2.27 make_int_polynomial_t		28
6.1.2.28 make_q32_t		28
6.1.2.29 make_q64_t		29
6.1.2.30 makefraction_t		29
6.1.2.31 mul_t		29
6.1.2.32 mulfractions_t	3	30
6.1.2.33 pi64	3	30
6.1.2.34 PI_fraction	3	30
6.1.2.35 pow_t	3	30
6.1.2.36 pq64	3	30
6.1.2.37 q32	3	31
6.1.2.38 q64	3	31
6.1.2.39 sin	3	31
6.1.2.40 sinh	3	31
6.1.2.41 SQRT2_fraction	3	31
6.1.2.42 SQRT3_fraction	3	32
6.1.2.43 stirling_1_signed_t	3	32
6.1.2.44 stirling_1_unsigned_t	3	32
6.1.2.45 stirling_2_t	3	32
6.1.2.46 sub_t	3	33
6.1.2.47 tan	3	33
6.1.2.48 tanh	3	33
6.1.2.49 taylor	3	33
6.1.2.50 vadd_t		34
6.1.2.51 vmul_t	3	34
6.1.3 Function Documentation	3	34
6.1.3.1 aligned_malloc()	(34

6.1.3.2 field()	34
6.1.4 Variable Documentation	35
6.1.4.1 alternate_v	35
6.1.4.2 bernoulli_v	35
6.1.4.3 combination_v	35
6.1.4.4 factorial_v	36
6.2 aerobus::internal Namespace Reference	36
6.2.1 Detailed Description	39
6.2.2 Typedef Documentation	39
6.2.2.1 make_index_sequence_reverse	39
6.2.2.2 type_at_t	40
6.2.3 Function Documentation	40
6.2.3.1 index_sequence_reverse()	40
6.2.4 Variable Documentation	40
6.2.4.1 is_instantiation_of_v	40
6.3 aerobus::known_polynomials Namespace Reference	40
6.3.1 Detailed Description	40
6.3.2 Enumeration Type Documentation	40
6.3.2.1 hermite_kind	40
7 Concept Documentation	41
7.1 aerobus::IsEuclideanDomain Concept Reference	41
7.1.1 Concept definition	41
7.1.2 Detailed Description	41
7.2 aerobus::IsField Concept Reference	41
7.2.1 Concept definition	41
7.2.2 Detailed Description	42
7.3 aerobus::IsRing Concept Reference	42
7.3.1 Concept definition	42
7.3.2 Detailed Description	42
8 Class Documentation	43
8.1 aerobus::polynomial < Ring >::val < coeffN >::coeff at < index, E > Struct Template Reference	
8.2 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0 index >	
0)>> Struct Template Reference	
8.2.1 Member Typedef Documentation	
8.2.1.1 type	
8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference	
8.3.1 Member Typedef Documentation	44
8.3.1.1 type	44
8.4 aerobus::ContinuedFraction < values > Struct Template Reference	. 44
8.4.1 Detailed Description	44

8.5 aerobus::ContinuedFraction< a0 > Struct Template Reference	5
8.5.1 Detailed Description	Ę
8.5.2 Member Typedef Documentation	Ę
8.5.2.1 type	Ę
8.5.3 Member Data Documentation	16
8.5.3.1 val	16
$8.6 \ aerobus:: Continued Fraction < a0, rest > Struct \ Template \ Reference \\ \ \ldots \\ \ \ldots \\ \ \ $	16
8.6.1 Detailed Description	16
8.6.2 Member Typedef Documentation	7
8.6.2.1 type	7
8.6.3 Member Data Documentation	7
8.6.3.1 val	7
8.7 aerobus::ConwayPolynomial Struct Reference	7
8.8 aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner< index, ghost > Struct Template Reference	17
8.8.1 Member Function Documentation	18
8.8.1.1 func()	18
8.9 aerobus::polynomial < Ring >::compensated_horner < arithmeticType, P >::EFTHorner <-1, ghost >	
Struct Template Reference	3-
8.9.1 Member Function Documentation	3-
8.9.1.1 func()	3.
8.10 aerobus::Embed $<$ Small, Large, E $>$ Struct Template Reference	Ę
8.10.1 Detailed Description	Ę
8.11 aerobus::Embed $<$ i32, i64 $>$ Struct Reference	Ę
8.11.1 Detailed Description	Ę
8.11.2 Member Typedef Documentation	Į
8.11.2.1 type	Ę
$8.12 \; aerobus:: Embed < polynomial < Small >, polynomial < Large > > Struct \; Template \; Reference \; \ldots \; 5 \; description \; Struct \; Template \; Struct \; Template \; Struct \; Template \; Struct \; Struct$	5(
8.12.1 Detailed Description	5(
8.12.2 Member Typedef Documentation	50
8.12.2.1 type	5(
8.13 aerobus::Embed $<$ q32, q64 $>$ Struct Reference	51
8.13.1 Detailed Description	51
8.13.2 Member Typedef Documentation	51
8.13.2.1 type	51
8.14 aerobus::Embed< Quotient< Ring, X >, Ring > Struct Template Reference	52
8.14.1 Detailed Description	52
8.14.2 Member Typedef Documentation	52
8.14.2.1 type	52
8.15 aerobus::Embed< Ring, FractionField< Ring > > Struct Template Reference	53
8.15.1 Detailed Description	5
8.15.2 Member Typedef Documentation	53
8.15.2.1 type	53

8.16 aerobus::Embed $<$ zpz $<$ x $>$, i32 $>$ Struct Template Reference	53
8.16.1 Detailed Description	54
8.16.2 Member Typedef Documentation	54
8.16.2.1 type	54
$8.17 \; aerobus::polynomial < Ring > ::horner_reduction_t < P > Struct \; Template \; Reference \; \ldots \; \ldots \; .$	54
8.17.1 Detailed Description	55
8.18 aerobus::i32 Struct Reference	55
8.18.1 Detailed Description	56
8.18.2 Member Typedef Documentation	57
8.18.2.1 add_t	57
8.18.2.2 div_t	57
8.18.2.3 eq_t	57
8.18.2.4 gcd_t	57
8.18.2.5 gt_t	58
8.18.2.6 inject_constant_t	58
8.18.2.7 inject_ring_t	58
8.18.2.8 inner_type	58
8.18.2.9 lt_t	58
8.18.2.10 mod_t	59
8.18.2.11 mul_t	59
8.18.2.12 one	59
8.18.2.13 pos_t	59
8.18.2.14 sub_t	60
8.18.2.15 zero	60
8.18.3 Member Data Documentation	60
8.18.3.1 eq_v	60
8.18.3.2 is_euclidean_domain	60
8.18.3.3 is_field	60
8.18.3.4 pos_v	61
8.19 aerobus::i64 Struct Reference	62
8.19.1 Detailed Description	63
8.19.2 Member Typedef Documentation	63
8.19.2.1 add_t	63
8.19.2.2 div_t	64
8.19.2.3 eq_t	64
8.19.2.4 gcd_t	64
8.19.2.5 gt_t	64
8.19.2.6 inject_constant_t	65
8.19.2.7 inject_ring_t	65
8.19.2.8 inner_type	65
8.19.2.9 lt_t	65
8.19.2.10 mod_t	66

8.19.2.11 mul_t	66
8.19.2.12 one	66
8.19.2.13 pos_t	66
8.19.2.14 sub_t	66
8.19.2.15 zero	67
8.19.3 Member Data Documentation	67
8.19.3.1 eq_v	67
8.19.3.2 gt_v	67
8.19.3.3 is_euclidean_domain	67
8.19.3.4 is_field	68
8.19.3.5 lt_v	68
8.19.3.6 pos_v	
8.20 aerobus::polynomial < Ring >::horner_reduction_t < P >::inner < index, stop > Struct Template Reference	
8.20.1 Member Typedef Documentation	69
8.20.1.1 type	69
8.21 aerobus::polynomial < Ring >::horner reduction t < P >::inner < stop, stop > Struct Template Ref-	
erence	69
8.21.1 Member Typedef Documentation	69
8.21.1.1 type	69
8.22 aerobus::is_prime $<$ n $>$ Struct Template Reference	69
8.22.1 Detailed Description	70
8.22.2 Member Data Documentation	. 70
8.22.2.1 value	. 70
8.23 aerobus::polynomial < Ring > Struct Template Reference	70
8.23.1 Detailed Description	72
8.23.2 Member Typedef Documentation	72
8.23.2.1 add_t	. 72
8.23.2.2 derive_t	. 72
8.23.2.3 div_t	. 73
8.23.2.4 eq_t	. 73
8.23.2.5 gcd_t	. 73
8.23.2.6 gt_t	. 73
8.23.2.7 inject_constant_t	. 74
8.23.2.8 inject_ring_t	. 74
8.23.2.9 lt_t	. 74
8.23.2.10 mod_t	. 74
8.23.2.11 monomial_t	75
8.23.2.12 mul_t	. 75
8.23.2.13 one	
8.23.2.14 pos_t	. 75
8.23.2.15 simplify_t	
8.23.2.16 sub_t	

8.23.2.17 X	77
8.23.2.18 zero	77
8.23.3 Member Data Documentation	78
8.23.3.1 is_euclidean_domain	78
8.23.3.2 is_field	78
8.23.3.3 pos_v	78
8.24 aerobus::type_list< Ts >::pop_front Struct Reference	78
8.24.1 Detailed Description	78
8.24.2 Member Typedef Documentation	79
8.24.2.1 tail	79
8.24.2.2 type	79
8.25 aerobus::Quotient $<$ Ring, X $>$ Struct Template Reference	79
8.25.1 Detailed Description	80
8.25.2 Member Typedef Documentation	80
8.25.2.1 add_t	80
8.25.2.2 div_t	81
8.25.2.3 eq_t	81
8.25.2.4 inject_constant_t	81
8.25.2.5 inject_ring_t	82
8.25.2.6 mod_t	82
8.25.2.7 mul_t	82
8.25.2.8 one	82
8.25.2.9 pos_t	83
8.25.2.10 zero	83
8.25.3 Member Data Documentation	83
8.25.3.1 eq_v	83
8.25.3.2 is_euclidean_domain	83
8.25.3.3 pos_v	83
8.26 aerobus::type_list< Ts >::split< index > Struct Template Reference	84
8.26.1 Detailed Description	84
8.26.2 Member Typedef Documentation	84
8.26.2.1 head	84
8.26.2.2 tail	84
8.27 aerobus::type_list< Ts > Struct Template Reference	85
8.27.1 Detailed Description	85
8.27.2 Member Typedef Documentation	86
8.27.2.1 at	86
8.27.2.2 concat	86
8.27.2.3 insert	86
8.27.2.4 push_back	87
8.27.2.5 push_front	87
8.27.2.6 remove	87

8.27.3 Member Data Documentation	87
8.27.3.1 length	87
8.28 aerobus::type_list<> Struct Reference	88
8.28.1 Detailed Description	88
8.28.2 Member Typedef Documentation	88
8.28.2.1 concat	88
8.28.2.2 insert	88
8.28.2.3 push_back	88
8.28.2.4 push_front	88
8.28.3 Member Data Documentation	89
8.28.3.1 length	89
8.29 aerobus::i32::val $<$ x $>$ Struct Template Reference	89
8.29.1 Detailed Description	89
8.29.2 Member Typedef Documentation	90
8.29.2.1 enclosing_type	90
8.29.2.2 is_zero_t	90
8.29.3 Member Function Documentation	90
8.29.3.1 get()	90
8.29.3.2 to_string()	90
8.29.4 Member Data Documentation	90
8.29.4.1 v	90
8.30 aerobus::i64::val < x > Struct Template Reference	91
8.30.1 Detailed Description	91
8.30.2 Member Typedef Documentation	92
8.30.2.1 enclosing_type	92
8.30.2.2 inner_type	92
8.30.2.3 is_zero_t	92
8.30.3 Member Function Documentation	92
8.30.3.1 get()	92
8.30.3.2 to_string()	92
8.30.4 Member Data Documentation	93
8.30.4.1 v	93
8.31 aerobus::polynomial< Ring >::val< coeffN, coeffs > Struct Template Reference	93
8.31.1 Detailed Description	94
8.31.2 Member Typedef Documentation	94
8.31.2.1 aN	94
8.31.2.2 coeff_at_t	94
8.31.2.3 enclosing_type	95
8.31.2.4 is_zero_t	95
8.31.2.5 ring_type	95
8.31.2.6 strip	95
8.31.2.7 value_at_t	95

8.31.3 Member Function Documentation	95
8.31.3.1 compensated_eval()	95
8.31.3.2 eval()	96
8.31.3.3 to_string()	96
8.31.4 Member Data Documentation	97
8.31.4.1 degree	97
8.31.4.2 is_zero_v	97
8.32 aerobus::Quotient $<$ Ring, $X>::$ val $<$ $V>$ Struct Template Reference	97
8.32.1 Detailed Description	97
8.32.2 Member Typedef Documentation	98
8.32.2.1 raw_t	98
8.32.2.2 type	98
8.33 aerobus::zpz::val< x > Struct Template Reference	98
8.33.1 Detailed Description	98
8.33.2 Member Typedef Documentation	99
8.33.2.1 enclosing_type	99
8.33.2.2 is_zero_t	99
8.33.3 Member Function Documentation	99
8.33.3.1 get()	99
8.33.3.2 to_string()	99
8.33.4 Member Data Documentation	00
8.33.4.1 is_zero_v	00
0.00.4.0.4	
8.33.4.2 v	00
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference	
	00
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference	00 01
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference	00 01 01
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference	00 01 01 01
8.34 aerobus::polynomial Ring >::val< coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1	00 01 01 01
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1	00 01 01 01 01
8.34 aerobus::polynomial Ring >::val< coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1 8.34.2.3 enclosing_type 1	00 01 01 01 01 01
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1 8.34.2.3 enclosing_type 1 8.34.2.4 is_zero_t 1	00 01 01 01 01 02 02
8.34 aerobus::polynomial Ring >::val< coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1 8.34.2.3 enclosing_type 1 8.34.2.4 is_zero_t 1 8.34.2.5 ring_type 1	00 01 01 01 01 02 02
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1 8.34.2.3 enclosing_type 1 8.34.2.4 is_zero_t 1 8.34.2.5 ring_type 1 8.34.2.6 strip 1	00 01 01 01 01 02 02 02
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1 8.34.2.3 enclosing_type 1 8.34.2.4 is_zero_t 1 8.34.2.5 ring_type 1 8.34.2.6 strip 1 8.34.2.7 value_at_t 1	00 01 01 01 02 02 02 02
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1 8.34.2.3 enclosing_type 1 8.34.2.4 is_zero_t 1 8.34.2.5 ring_type 1 8.34.2.6 strip 1 8.34.2.7 value_at_t 1 8.34.3 Member Function Documentation 1	00 01 01 01 02 02 02 02
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1 8.34.2.3 enclosing_type 1 8.34.2.4 is_zero_t 1 8.34.2.5 ring_type 1 8.34.2.6 strip 1 8.34.2.7 value_at_t 1 8.34.3 Member Function Documentation 1 8.34.3.1 compensated_eval() 1	00 01 01 01 02 02 02 02 02
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1 8.34.2.3 enclosing_type 1 8.34.2.4 is_zero_t 1 8.34.2.5 ring_type 1 8.34.2.6 strip 1 8.34.2.7 value_at_t 1 8.34.3 Member Function Documentation 1 8.34.3.1 compensated_eval() 1 8.34.3.2 eval() 1	00 01 01 01 02 02 02 02 02 02 03
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1 8.34.2.3 enclosing_type 1 8.34.2.4 is_zero_t 1 8.34.2.5 ring_type 1 8.34.2.6 strip 1 8.34.2.7 value_at_t 1 8.34.3 Member Function Documentation 1 8.34.3.1 compensated_eval() 1 8.34.3.2 eval() 1 8.34.3.3 to_string() 1	00 01 01 01 02 02 02 02 02 03
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1 8.34.2.3 enclosing_type 1 8.34.2.4 is_zero_t 1 8.34.2.5 ring_type 1 8.34.2.6 strip 1 8.34.2.7 value_at_t 1 8.34.3 Member Function Documentation 1 8.34.3.1 compensated_eval() 1 8.34.3.2 eval() 1 8.34.3.3 to_string() 1 8.34.4 Member Data Documentation 1	00 01 01 01 02 02 02 02 02 03 03
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference 1 8.34.1 Detailed Description 1 8.34.2 Member Typedef Documentation 1 8.34.2.1 aN 1 8.34.2.2 coeff_at_t 1 8.34.2.3 enclosing_type 1 8.34.2.4 is_zero_t 1 8.34.2.5 ring_type 1 8.34.2.6 strip 1 8.34.2.7 value_at_t 1 8.34.3 Member Function Documentation 1 8.34.3.1 compensated_eval() 1 8.34.3.2 eval() 1 8.34.3.3 to_string() 1 8.34.4.4 Member Data Documentation 1 8.34.4.1 degree 1	00 01 01 01 02 02 02 02 03 03 03

8.35.2 Member Typedef Documentation	. 105
8.35.2.1 add_t	. 105
8.35.2.2 div_t	. 105
8.35.2.3 eq_t	. 106
8.35.2.4 gcd_t	. 106
8.35.2.5 gt_t	. 106
8.35.2.6 inject_constant_t	. 107
8.35.2.7 inner_type	. 107
8.35.2.8 lt_t	. 107
8.35.2.9 mod_t	. 107
8.35.2.10 mul_t	. 108
8.35.2.11 one	. 108
8.35.2.12 pos_t	. 108
8.35.2.13 sub_t	. 108
8.35.2.14 zero	. 109
8.35.3 Member Data Documentation	. 109
8.35.3.1 eq_v	. 109
8.35.3.2 gt_v	. 109
8.35.3.3 is_euclidean_domain	. 109
8.35.3.4 is_field	. 109
8.35.3.5 lt_v	. 110
8.35.3.6 pos_v	. 110
9 File Documentation	111
9.1 README.md File Reference	
9.2 src/aerobus.h File Reference	
9.3 aerobus.h	
9.4 src/examples.h File Reference	. 204
9.5 examples.h	_
3.3 examples.ii	. 204
10 Examples	205
10.1 examples/hermite.cpp	. 205
10.2 examples/custom_taylor.cpp	. 205
10.3 examples/fp16.cu	. 206
10.4 examples/continued_fractions.cpp	. 206
10.5 examples/modular_arithmetic.cpp	. 207
10.6 examples/make_polynomial.cpp	. 207
10.7 examples/polynomials_over_finite_field.cpp	. 208
10.8 examples/compensated_horner.cpp	. 208
Index	211
	_

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 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. And Google's Benchmark library. Then move to top directory:

```
rm -rf build
mkdir build
cd build
cmake ..
make benchmarks
./benchmarks
```

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
- $\bullet \ \, \mathrm{sub_t}: a \ type \ (specialization \ of \ val), \ representing \ subtraction \ between \ two \ values$
- 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 Structures 3

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.

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
- · laquerre
- hermite_prob
- hermite_phys
- bernstein
- legendre
- bernoulli

1.2.4 Conway polynomials

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

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

4 Introduction

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
- $expm1 e^x 1$
- lnp1 ln(x+1)
- geom $\frac{1}{1-x}$
- sin
- cos
- tan
- sh
- cosh
- tanh
- asin
- acos
- acosh
- asinh
- atanh

Having the capacity of specifying the degree is very important, as users may use other formats than float64 or float32 which require higher or lower degree to achieve correct or acceptable precision.

It's possible to define Taylor expansion by implementing a $coeff_at$ structure which must meet the following requirement:

- Being template in Integers (typename) and index (size_t);
- Exposing a type alias type, some specialization of FractionField<Integers>::val.

1.3 Operations 5

For example, to define the serie $1 + x + x^2 + x^3 + \ldots$, 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:

```
ompute_expm1(unsigned long, double const*, double*):
          rax, [rdi-1]
  cmp
          rax, 2
  ibe
          .L5
 mov
          rcx, rdi
          eax, eax
  vxorpd xmm1, xmm1, xmm1
 vbroadcastsd ymm14, QWORD PTR .LC1[rip]
vbroadcastsd ymm13, QWORD PTR .LC3[rip]
shr rcx, 2
 vbroadcastsd ymm12, QWORD PTR .LC5[rip] vbroadcastsd ymm11, QWORD PTR .LC7[rip]
          rcx, 5
  vbroadcastsd ymm10, QWORD PTR .LC9[rip]
 vbroadcastsd
                   ymm9, QWORD PTR .LC11[rip]
 vbroadcastsd ymm8, QWORD PTR .LC13[rip] vbroadcastsd ymm7, QWORD PTR .LC15[rip]
  vbroadcastsd
                  ymm6, QWORD PTR .LC17[rip]
 vbroadcastsd
vbroadcastsd
                   ymm5, QWORD PTR .LC19[rip]
                   ymm4, QWORD PTR .LC21[rip]
 vbroadcastsd
                  ymm3, QWORD PTR .LC23[rip]
  vbroadcastsd
                   ymm2, QWORD PTR .LC25[rip]
.L3:
  vmovupd ymm15, YMMWORD PTR [rsi+rax]
  vmovapd ymm0, ymm15
  vfmadd132pd
                   ymm0, ymm14, ymm1
 vfmadd132pd
                   ymm0, ymm13, ymm15
  vfmadd132pd
                   ymm0, ymm12, ymm15
  vfmadd132pd
                   ymm0, ymm11, ymm15
  vfmadd132pd
                   ymm0, ymm10, ymm15
  vfmadd132pd
                   ymm0, ymm9, ymm15
                   ymm0, ymm8, ymm15
  vfmadd132pd
 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
 add
  cmp
          rcx, rax
  jne
          .L3
          rax, rdi
  and
          rax,
 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.

6 Introduction

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 $\ \,$ quotient $\ \,$ ring R/X where X is a principal ideal generated by some element, as we know this kind of ideal is two-sided as long as R is commutative (and we assume it is).

For example, if we want R to be \mathbb{Z} represented as aerobus::i64, we can express arithmetic modulo 17 using: using namespace aerobus; using ZpZ = Quotient < i64, i64::val < 17 >>;

As we could have using zpz<17>.

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

1.4 Misc

1.4.1 Continued Fractions

Aerobus gives an implementation for continued fractions. It can be used this way: using namespace aerobus; using T = ContinuedFraction<1,2,3,4>; constexpr double x = T::val;

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

1.5 CUDA

When compiled with nvcc and the flag WITH_CUDA_FP16, Aerobus provides some kind of support of 16 bits integers and floats (aka $__half$).

Unfortunately, NVIDIA did not put enough constexpr in its <code>cuda_fp16.h</code> header, so we had to implement our own constexpr static_cast from int16_t to <code>__half</code> to make integers polynomials work with <code>__half</code>. See <code>thisbug</code>.

More, it's (at this time), not possible to make it work for __half2 because of another bug.

Please push to make these bug fixed by NVIDIA.

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

aerobus	
Main namespace for all publicly exposed types or functions	15
aerobus::internal	
Internal implementations, subject to breaking changes without notice	36
aerobus::known_polynomials	
Families of well known polynomials such as Hermite or Bernstein	40

8 Namespace Index

Concept Index

3.1 Concepts

Here is a list of all concepts with brief descriptions:

aerobus::IsEuclideanDomain	
Concept to express R is an euclidean domain	41
aerobus::IsField	
Concept to express R is a field	41
aerobus::IsRing	
Concept to express R is a Ring	42

10 Concept Index

Class Index

4.1 Class List

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

aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, E >	43
aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable_if_t < (index < $0 index > 0 >$ 43	
aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)>>	44
aerobus::ContinuedFraction < values >	
Continued fraction a0 + $\frac{1}{a_1 + \frac{1}{a_2 + \dots}}$	44
aerobus::ContinuedFraction $<$ a0 $>$	
Specialization for only one coefficient, technically just 'a0'	45
aerobus::ContinuedFraction< a0, rest >	
Specialization for multiple coefficients (strictly more than one)	46
aerobus::ConwayPolynomial	47
aerobus::polynomial < Ring >::compensated_horner < arithmeticType, P >::EFTHorner < index, ghost >	47
aerobus::polynomial < Ring >::compensated_horner < arithmeticType, P >::EFTHorner <-1, ghost >	48
aerobus::Embed < Small, Large, E >	
Embedding - struct forward declaration	49
aerobus::Embed < i32, i64 >	
Embeds i32 into i64	49
aerobus::Embed< polynomial< Small >, polynomial< Large >>	
Embeds polynomial < Small > into polynomial < Large >	50
aerobus::Embed< q32, q64 >	
Embeds q32 into q64	51
aerobus::Embed< Quotient< Ring, X >, Ring >	
Embeds Quotient <ring, x=""> into Ring</ring,>	52
aerobus::Embed< Ring, FractionField< Ring >>	
Embeds values from Ring to its field of fractions	53
aerobus::Embed< zpz< x >, i32 >	
Embeds zpz values into i32	53
aerobus::polynomial< Ring >::horner_reduction_t< P >	
Used to evaluate polynomials over a value in Ring	54
aerobus::i32	
32 bits signed integers, seen as a algebraic ring with related operations	55
aerobus::i64	
64 bits signed integers, seen as a algebraic ring with related operations	62
aerobus::polynomial < Ring >::horner_reduction_t < P >::inner < index, stop >	68
aerobus::polynomial < Ring >::horner_reduction_t < P >::inner < stop, stop >	69

12 Class Index

aerobus::is_prime< n >	
Checks if n is prime	. 69
aerobus::polynomial < Ring >	. 70
aerobus::type_list< Ts >::pop_front	
Removes types from head of the list	. 78
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	. 79
aerobus::type_list< Ts >::split< index >	
Splits list at index	. 84
aerobus::type_list< Ts >	
Empty pure template struct to handle type list	. 85
aerobus::type_list<>	
Specialization for empty type list	. 88
aerobus::i32::val< x >	
Values in i32, again represented as types	. 89
aerobus::i64::val < x >	
Values in i64	. 91
aerobus::polynomial < Ring >::val < coeffN, coeffs >	
Values (seen as types) in polynomial ring	. 93
aerobus::Quotient < Ring, X >::val < V >	
Projection values in the quotient ring	. 97
aerobus::zpz::val< x >	
Values in zpz	. 98
aerobus::polynomial< Ring >::val< coeffN >	
Specialization for constants	. 100
aerobus::zpz	
Congruence classes of integers modulo n (32 hits)	103

File Index

5.1 File List

Here is a list of all files with brief descriptions:

src/aerobus.h .						 					 											11	11
src/examples.h						 	 				 											20)4

14 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 ←
  _type::template sub_t< typename val::enclosing_type::zero, val > >
     computes absolute value of 'val' val must be a 'value' in a Ring satisfying 'lsEuclideanDomain' concept

    template<typename Ring >

  using FractionField = typename internal::FractionFieldImpl< Ring >::type
      Fraction field of an euclidean domain, such as Q for Z.
• template<typename X , typename Y>
  using add_t = typename X::enclosing_type::template add_t < X, Y >
     generic addition
• template<typename X , typename Y>
```

using sub_t = typename X::enclosing_type::template sub_t < X, Y >

```
generic subtraction
• template<typename X , typename Y >
  using mul_t = typename X::enclosing_type::template mul_t < X, Y >
     generic multiplication

    template<typename X , typename Y >

  using div_t = typename X::enclosing_type::template div_t < X, Y >
     generic division

 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

 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
     Rell 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_1_signed_t = typename internal::stirling_1_helper< T, n, k >::type
      Stirling number of first king (signed) - as types.
• template<typename T , int n, int k>
  using stirling_1_unsigned_t = abs_t< typename internal::stirling_1_helper< T, n, k >::type >
      Stirling number of first king (unsigned) - as types.
• template<typename T , int n, int k>
  using stirling 2 t = typename internal::stirling 2 helper< T, n, k >::type
      Stirling number of second king – 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 >
• template<typename Integers , size_t deg>
  using cosh = taylor < Integers, internal::cosh coeff, deg >
     \cosh(x) hyperbolic cosine
• template<typename Integers , size_t deg>
  using cos = taylor < Integers, internal::cos coeff, deg >
     cos(x) cosinus

    template<typename Integers , size_t deg>

  using geometric_sum = taylor< Integers, internal::geom_coeff, deg >
```

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

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

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

    using SQRT3 fraction = ContinuedFraction
    1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
     1, 2, 1, 2, 1, 2 >
               approximation of
```

Functions

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

Variables

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 add_t

```
template<typename X , typename Y >
using aerobus::add_t = typedef typename X::enclosing_type::template add_t<X, Y>
```

generic addition

Template Parameters

X	a value in a ring providing add_t operator
Y	a value in same ring

6.1.2.3 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

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

6.1.2.4 alternate_t

```
template<typename T , int k> using aerobus::alternate_t = typedef typename internal::alternate<T, k>::type (-1)^k as type in T
```

Template Parameters

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

6.1.2.5 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

6.1.2.6 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

6.1.2.7 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

6.1.2.8 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

6.1.2.9 bell_t

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

Bell numbers.

Template Parameters

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

6.1.2.10 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	

6.1.2.11 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)
```

6.1.2.12 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

6.1.2.13 cosh

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

Template Parameters

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

6.1.2.14 div_t

```
template<typename X , typename Y >
using aerobus::div_t = typedef typename X::enclosing_type::template div_t<X, Y>
```

generic division

Template Parameters

Χ	a value in a a euclidean domain
Y	a value in same Euclidean domain

6.1.2.15 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 > 0
```

approximation of \boldsymbol{e}

6.1.2.16 embed_int_poly_in_fractions_t

embed a polynomial with integers coefficients into rational coefficients polynomials

Lives in polynomial<FractionField<Ring>>

Template Parameters

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

6.1.2.17 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

6.1.2.18 expm1

```
template<typename Integers , size_t deg> using aerobus::expml = typedef typename polynomial<FractionField<Integers>>::template sub_t<exp<Integers, deg>, typename polynomial<FractionField<Integers>>::one> e^x-1
```

Template Parameters

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

6.1.2.19 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	

6.1.2.20 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)

6.1.2.21 fpq64

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

polynomial with 64 bits rational coefficients

6.1.2.22 FractionField

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

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

Template Parameters

```
Ring
```

6.1.2.23 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)
```

6.1.2.24 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

6.1.2.25 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

6.1.2.26 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

6.1.2.27 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

6.1.2.28 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

6.1.2.29 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

6.1.2.30 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

6.1.2.31 mul_t

```
template<typename X , typename Y >
using aerobus::mul_t = typedef typename X::enclosing_type::template mul_t<X, Y>
```

generic multiplication

Template Parameters

Χ	a value in a ring providing mul_t operator
Y	a value in same ring

6.1.2.32 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>

6.1.2.33 pi64

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

polynomial with 64 bits integers coefficients

6.1.2.34 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>
```

representation of π as a continued fraction

6.1.2.35 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

6.1.2.36 pq64

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

polynomial with 64 bits rationals coefficients

6.1.2.37 q32

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

32 bits rationals rationals with 32 bits numerator and denominator

6.1.2.38 q64

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

64 bits rationals rationals with 64 bits numerator and denominator

6.1.2.39 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

6.1.2.40 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

6.1.2.41 SQRT2_fraction

approximation of $\sqrt{2}$

6.1.2.42 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

6.1.2.43 stirling_1_signed_t

```
template<typename T , int n, int k> using aerobus::stirling_1_signed_t = typedef typename internal::stirling_1_helper<T, n, k> \leftarrow ::type
```

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

Template Parameters

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

6.1.2.44 stirling_1_unsigned_t

```
template<typename T , int n, int k>
using aerobus::stirling_1_unsigned_t = typedef abs_t<typename internal::stirling_1_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)

6.1.2.45 stirling_2_t

```
\label{template} $$ template < typename T , int n, int k > $$ using $$ aerobus::stirling_2_t = typedef typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::type $$ typename internal::stirling_2_helper < T, n, k > ::ty
```

Stirling number of second king – as types.

Template Parameters

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

6.1.2.46 sub_t

```
template<typename X , typename Y >
using aerobus::sub_t = typedef typename X::enclosing_type::template sub_t<X, Y>
```

generic subtraction

Template Parameters

Χ	a value in a ring providing sub_t operator	
Y	a value in same ring	

6.1.2.47 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

6.1.2.48 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

6.1.2.49 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	

Generated by Doxygen

6.1.2.50 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
```

6.1.2.51 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)[^]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 asinh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >>

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 _{\mbox{is\_prime}}< 2, i >

    struct _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 AbelHelper

• struct AllOneHelper

    struct AllOneHelper< 0, I >

· 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 atan_coeff

    struct atan_coeff_helper

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

    struct atanh_coeff_helper

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

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

    struct bell_helper

• struct bell_helper< T, 0 >

    struct bell_helper< T, 1 >

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

    struct bernoulli

    struct bernoulli < T, 0 >

• struct bernoulli_coeff
• struct bernoulli helper

    struct bernoulli_helper< T, accum, m, m >

    struct bernstein helper

    struct bernstein_helper< 0, 0, I >

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

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

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

• struct BesselHelper

    struct BesselHelper< 0, I >

    struct BesselHelper< 1, I >

    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 FloatLayout

    struct FloatLayout< double >

    struct FloatLayout< float >

    struct fma helper

    struct fma_helper< double >

struct fma_helper< float >

    struct fma_helper< int16_t >

    struct fma_helper< int32_t >
```

struct fma_helper< int64_t > struct FractionFieldImpl struct FractionFieldImpl< Field, std::enable_if_t< Field::is_field >> struct FractionFieldImpl< Ring, std::enable_if_t<!Ring::is_field >> · struct gcd greatest common divisor computes the greatest common divisor exposes it in gcd<A, B>::type as long as Ring type is an integral domain struct gcd< Ring, std::enable_if_t< Ring::is_euclidean_domain > > · struct geom coeff · struct hermite helper struct hermite helper< 0, known polynomials::hermite kind::physicist, I > struct hermite_helper< 0, known_polynomials::hermite_kind::probabilist, I > struct hermite_helper< 1, known_polynomials::hermite_kind::physicist, I > struct hermite_helper< 1, known_polynomials::hermite_kind::probabilist, I > struct hermite_helper< deg, known_polynomials::hermite_kind::physicist, l > struct hermite helper< deg, known polynomials::hermite kind::probabilist, l > · struct insert_h · struct is instantiation of struct is_instantiation_of< TT, TT< Ts... >> • struct laguerre helper • struct laquerre 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 staticcast • struct stirling 1 helper struct stirling_1_helper< T, 0, 0 > struct stirling 1_helper< T, 0, n, std::enable_if_t<(n > 0)>> struct stirling_1_helper< T, n, 0, std::enable_if_t<(n > 0)>> struct stirling_1_helper< T, n, k, std::enable_if_t<(k > 0) &&(n > 0)> > • struct stirling 2 helper

struct stirling_2_helper< T, 0, n, std::enable_if_t<(n > 0)> >

- struct stirling_2_helper< T, n, 0, std::enable_if_t<(n > 0)> >
- struct stirling_2_helper< T, n, k, std::enable_if_t<(k > 0) &&(n > 0) &&(k < n)> >
- struct stirling_2_helper< T, n, n, std::enable_if_t<(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 touchard coeff
- 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</li>
```

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

Enumerations

enum hermite_kind { probabilist , physicist }

6.3.1 Detailed Description

families of well known polynomials such as Hermite or Bernstein

6.3.2 Enumeration Type Documentation

6.3.2.1 hermite_kind

enum aerobus::known_polynomials::hermite_kind

Enumerator

probabilist	
physicist	

Chapter 7

Concept Documentation

7.1 aerobus::IsEuclideanDomain Concept Reference

Concept to express R is an euclidean domain.

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

7.1.1 Concept definition

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

7.1.2 Detailed Description

Concept to express R is an euclidean domain.

7.2 aerobus::IsField Concept Reference

Concept to express R is a field.

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

7.2.1 Concept definition

7.2.2 Detailed Description

Concept to express R is a field.

7.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring.

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

7.3.1 Concept definition

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

7.3.2 Detailed Description

Concept to express R is a Ring.

Chapter 8

Class Documentation

8.1 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E > Struct Template Reference

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

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

- src/aerobus.h
- 8.2 aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable_if_t < (index < 0||index > 0) > > Struct Template Reference

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

Public Types

• using type = typename Ring::zero

8.2.1 Member Typedef Documentation

8.2.1.1 type

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

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

• src/aerobus.h

8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference

#include <aerobus.h>

Public Types

using type = aN

8.3.1 Member Typedef Documentation

8.3.1.1 type

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

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

• src/aerobus.h

8.4 aerobus::ContinuedFraction < values > Struct Template Reference

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

Examples

examples/continued_fractions.cpp.

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

• src/aerobus.h

8.5 aerobus::ContinuedFraction < a0 > Struct Template Reference

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

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

Public Types

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

Static Public Attributes

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

8.5.1 Detailed Description

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

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

Template Parameters

a0	an integer
	int64_t

8.5.2 Member Typedef Documentation

8.5.2.1 type

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

represented value as aerobus::q64

8.5.3 Member Data Documentation

8.5.3.1 val

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

represented value as double

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

· src/aerobus.h

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

specialization for multiple coefficients (strictly more than one)

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

Public Types

using type = q64::template add_t< typename q64::template inject_constant_t< a0 >, typename q64
 ::template div_t< typename q64::one, typename ContinuedFraction< rest... >::type > >
 represented value as aerobus::q64

Static Public Attributes

static constexpr double val = type::template get<double>()
 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::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner< index, ghost > Struct Template Reference

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

Static Public Member Functions

• static INLINED void func (arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType *r)

8.8.1 Member Function Documentation

8.8.1.1 func()

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

· src/aerobus.h

8.9 aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner<-1, ghost > Struct Template Reference

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

Static Public Member Functions

static INLINED DEVICE void func (arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmetic
 —
 Type *r)

8.9.1 Member Function Documentation

8.9.1.1 func()

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

• src/aerobus.h

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

embedding - struct forward declaration

8.10.1 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

8.11 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
```

8.11.1 Detailed Description

embeds i32 into i64

8.11.2 Member Typedef Documentation

8.11.2.1 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

8.12 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

8.12.1 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

8.12.2 Member Typedef Documentation

8.12.2.1 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

8.13 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
```

8.13.1 Detailed Description

embeds q32 into q64

8.13.2 Member Typedef Documentation

8.13.2.1 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

8.14 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.
```

8.14.1 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

8.14.2 Member Typedef Documentation

8.14.2.1 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

8.15 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.
```

8.15.1 Detailed Description

```
template<typename Ring> struct aerobus::Embed< Ring, FractionField< Ring > >
```

embeds values from Ring to its field of fractions

Template Parameters

Ring an integers ring, such as i32

8.15.2 Member Typedef Documentation

8.15.2.1 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

8.16 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
```

8.16.1 Detailed Description

8.16.2 Member Typedef Documentation

8.16.2.1 type

an integer

```
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

8.17 aerobus::polynomial< Ring >::horner_reduction_t< P > Struct Template Reference

Used to evaluate polynomials over a value in Ring.

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

Classes

- struct inner
- struct inner< stop, stop >

8.17.1 Detailed Description

```
template<typename Ring>
template<typename P>
struct aerobus::polynomial< Ring >::horner_reduction_t< P >
```

Used to evaluate polynomials over a value in Ring.

Template Parameters

```
P a value in polynomial < Ring >
```

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

· src/aerobus.h

8.18 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

```
• template<typename v1 , typename v2 >
  using sub_t = typename sub< v1, v2 >::type
     substraction operator yields v1 - v2

    template<typename v1 , typename v2 >

  using mul_t = typename mul < v1, v2 >::type
      multiplication operator yields v1 * v2

    template<typename v1 , typename v2 >

  using div_t = typename div < v1, v2 >::type
      division operator yields v1 / v2
• template<typename v1 , typename v2 >
  using mod_t = typename remainder < v1, v2 >::type
      modulus operator yields v1 % v2

    template<typename v1 , typename v2 >

  using gt t = typename gt < v1, v2 > ::type
      strictly greater operator (v1 > v2) yields v1 > v2
• template<typename v1 , typename v2 >
  using lt_t = typename lt< v1, v2 >::type
     strict less operator (v1 < v2) yields v1 < v2

    template<typename v1 , typename v2 >

  using eq_t = typename eq< v1, v2 >::type
      equality operator (type) yields v1 == v2 as std::integral_constant<bool>

    template<typename v1 , typename v2 >

  using gcd_t = gcd_t < i32, v1, v2 >
     greatest common divisor yields GCD(v1, v2)
• template<typename v >
  using pos_t = typename pos< v >::type
     positivity operator yields v > 0 as std::true type or std::false type
```

Static Public Attributes

```
    static constexpr bool is_field = false
        integers are not a field
    static constexpr bool is_euclidean_domain = true
        integers are an euclidean domain
    template<typename v1, typename v2 >
        static constexpr bool eq_v = eq_t<v1, v2>::value
        equality operator (boolean value)
    template<typename v >
        static constexpr bool pos_v = pos_t<v>::value
        positivity (boolean value) yields v > 0 as boolean value
```

8.18.1 Detailed Description

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

Examples

examples/compensated_horner.cpp.

8.18.2 Member Typedef Documentation

8.18.2.1 add t

```
template<typename v1 , typename v2 >
using aerobus::i32::add_t = typename add<v1, v2>::type
```

addition operator yields v1 + v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.2 div_t

```
template<typename v1 , typename v2 >
using aerobus::i32::div_t = typename div<v1, v2>::type
```

division operator yields v1 / v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.3 eq_t

```
template<typename v1 , typename v2 >
using aerobus::i32::eq_t = typename eq<v1, v2>::type
```

equality operator (type) yields v1 == v2 as std::integral_constant<bool>

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.4 gcd_t

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

greatest common divisor yields GCD(v1, v2)

Template Parameters

v1	a value in i <mark>32</mark>
v2	a value in i32

8.18.2.5 gt t

```
template<typename v1 , typename v2 >
using aerobus::i32::gt_t = typename gt<v1, v2>::type
```

strictly greater operator (v1 > v2) yields v1 > v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.6 inject_constant_t

```
template<auto x>
using aerobus::i32::inject_constant_t = val<static_cast<int32_t>(x)>
```

inject a native constant

Template Parameters



8.18.2.7 inject_ring_t

```
template<typename v >
using aerobus::i32::inject_ring_t = v
```

8.18.2.8 inner_type

```
using aerobus::i32::inner_type = int32_t
```

8.18.2.9 It t

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

strict less operator (v1 < v2) yields v1 < v2

Template Parameters

v1	a value in i <mark>32</mark>
v2	a value in i32

8.18.2.10 mod_t

```
template<typename v1 , typename v2 >
using aerobus::i32::mod_t = typename remainder<v1, v2>::type
```

modulus operator yields v1 % v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.11 mul_t

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

multiplication operator yields v1 * v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.12 one

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

constant one

8.18.2.13 pos_t

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

positivity operator yields v>0 as std::true_type or std::false_type

Template Parameters

```
v a value in i32
```

8.18.2.14 sub_t

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

substraction operator yields v1 - v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.15 zero

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

constant zero

8.18.3 Member Data Documentation

8.18.3.1 eq_v

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

equality operator (boolean value)

Template Parameters

v1	
v2	

8.18.3.2 is_euclidean_domain

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

integers are an euclidean domain

8.18.3.3 is_field

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

integers are not a field

8.18.3.4 pos_v

```
template<typename v >
constexpr bool aerobus::i32::pos_v = pos_t < v > ::value [static], [constexpr]
```

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

Template Parameters

```
v a value in i32
```

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

• src/aerobus.h

8.19 aerobus::i64 Struct Reference

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

using mod_t = typename remainder < v1, v2 >::type

```
#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)>
     injects constant as an i64 value

    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
     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 integer division
• template<typename v1 , typename v2 >
```

```
modulus operator
• template<typename v1 , typename v2 >
  using gt_t = typename gt < v1, v2 >::type
      strictly greater operator yields v1 > v2 as std::true type or std::false type
• template<typename v1, typename v2 >
  using It_t = typename It< v1, v2 >::type
     strict less operator yields v1 < v2 as std::true_type or std::false_type
• template<typename v1 , typename v2 >
  using eq_t = typename eq< v1, v2 >::type
      equality operator yields v1 == v2 as std::true_type or std::false_type
• template<typename v1 , typename v2 >
  using gcd_t = gcd_t < i64, v1, v2 >
     greatest common divisor yields GCD(v1, v2) as instanciation of i64::val

    template<typename v >

  using pos_t = typename pos< v >::type
     is v posititive yields v > 0 as std::true_type or std::false_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
            strictly smaller operator yields v1 < v2 as boolean value</li>
    template<typename v1, typename v2 >
        static constexpr bool eq_v = eq_t<v1, v2>::value
```

equality operator yields v1 == v2 as boolean value

static constexpr bool pos_v = pos_t<v>::value positivity yields v > 0 as boolean value

8.19.1 Detailed Description

template<typename v >

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

8.19.2 Member Typedef Documentation

8.19.2.1 add t

```
template<typename v1 , typename v2 >
using aerobus::i64::add_t = typename add<v1, v2>::type
addition operator
```

Template Parameters

v1	: an element of aerobus::i64::val	
v2	: an element of aerobus::i64::val	

8.19.2.2 div_t

```
template<typename v1 , typename v2 >
using aerobus::i64::div_t = typename div<v1, v2>::type
```

division operator integer division

Template Parameters

v1	: an element of aerobus::i64::val
v2	: an element of aerobus::i64::val

8.19.2.3 eq_t

```
template<typename v1 , typename v2 >
using aerobus::i64::eq_t = typename eq<v1, v2>::type
```

equality 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

8.19.2.4 gcd_t

```
template<typename v1 , typename v2 >
using aerobus::i64::gcd_t = gcd_t < i64, v1, v2>
```

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

8.19.2.5 gt_t

```
template<typename v1 , typename v2 >
using aerobus::i64::gt_t = typename gt<v1, v2>::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

8.19.2.6 inject_constant_t

```
template<auto x>
using aerobus::i64::inject_constant_t = val<static_cast<int64_t>(x)>
```

injects constant as an i64 value

Template Parameters



8.19.2.7 inject_ring_t

```
template<typename v >
using aerobus::i64::inject_ring_t = v
```

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

Template Parameters

```
v a value in i64
```

8.19.2.8 inner_type

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

type of represented values

8.19.2.9 lt_t

```
template<typename v1 , typename v2 >
using aerobus::i64::lt_t = typename lt<v1, v2>::type
```

strict less 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

8.19.2.10 mod_t

```
template<typename v1 , typename v2 >
using aerobus::i64::mod_t = typename remainder<v1, v2>::type
```

modulus operator

Template Parameters

```
v1 : an element of aerobus::i64::valv2 : an element of aerobus::i64::val
```

8.19.2.11 mul_t

```
template<typename v1 , typename v2 >
using aerobus::i64::mul_t = typename mul<v1, v2>::type
```

multiplication operator

Template Parameters

v1	: an element of aerobus::i64::val
v2	: an element of aerobus::i64::val

8.19.2.12 one

```
using aerobus::i64::one = val<1>
```

constant one

8.19.2.13 pos_t

```
template<typename v >
using aerobus::i64::pos_t = typename pos<v>::type
```

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

Template Parameters

```
v1 : an element of aerobus::i64::val
```

8.19.2.14 sub_t

```
template<typename v1 , typename v2 >
using aerobus::i64::sub_t = typename sub<v1, v2>::type
```

substraction operator

Template Parameters

v1	: an element of aerobus::i64::val
v2	: an element of aerobus::i64::val

8.19.2.15 zero

```
using aerobus::i64::zero = val<0>
```

constant zero

8.19.3 Member Data Documentation

8.19.3.1 eq_v

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

equality operator yields v1 == v2 as boolean value

Template Parameters

v1	: an element of aerobus::i64::val
v2	: an element of aerobus::i64::val

8.19.3.2 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

8.19.3.3 is_euclidean_domain

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

integers are an euclidean domain

8.19.3.4 is_field

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

integers are not a field

8.19.3.5 It v

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

strictly smaller operator yields v1 < v2 as boolean value

Template Parameters

v1	: an element of aerobus::i64::va	
v2	: an element of aerobus::i64::val	

8.19.3.6 pos_v

```
template<typename v >
constexpr bool aerobus::i64::pos_v = pos_t < v > ::value [static], [constexpr]
```

positivity yields v>0 as boolean value

Template Parameters

```
v : an element of aerobus::i64::val
```

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

• src/aerobus.h

8.20 aerobus::polynomial < Ring >::horner_reduction_t < P >::inner < index, stop > Struct Template Reference

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

Public Types

• template<typename accum , typename x > using type = typename horner_reduction_t< P >::template inner< index+1, stop > ::template type< typename Ring::template add_t< typename Ring::template mul_t< x, accum >, typename P::template coeff_ \leftarrow at_t< P::degree - index > >, x >

8.20.1 Member Typedef Documentation

8.20.1.1 type

```
template<typename Ring >
template<typename P >
template<size_t index, size_t stop>
template<typename accum , typename x >
using aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop >::type =
typename horner_reduction_t<P>::template inner<index + 1, stop> ::template type< typename
Ring::template add_t< typename Ring::template mul_t<x, accum>, typename P::template coeff_\top
at_t<P::degree - index> >, x>
```

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

• src/aerobus.h

8.21 aerobus::polynomial < Ring >::horner_reduction_t < P >::inner < stop, stop > Struct Template Reference

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

Public Types

```
    template<typename accum, typename x > using type = accum
```

8.21.1 Member Typedef Documentation

8.21.1.1 type

```
template<typename Ring >
template<typename P >
template<size_t stop>
template<typename accum , typename x >
using aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< stop, stop >::type =
accum
```

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

• src/aerobus.h

8.22 $aerobus::is_prime < n > Struct Template Reference$

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

8.22.1 Detailed Description

```
template < size_t n > struct aerobus::is_prime < n > checks if n is prime

Template Parameters
```

8.22.2 Member Data Documentation

8.22.2.1 value

```
template<size_t n>
constexpr bool aerobus::is_prime< n >::value = internal::_is_prime<n, 5>::value [static],
[constexpr]
```

true iff n is prime

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

• src/aerobus.h

8.23 aerobus::polynomial < Ring > Struct Template Reference

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

Classes

• struct horner_reduction_t

Used to evaluate polynomials over a value in Ring.

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 It_t = typename It_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 > >
     makes the constant (native type) polynomial a_0

    template<typename v >

  using inject_ring_t = val< v >
     makes the constant (ring type) polynomial a_0
```

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
```

8.23.1 Detailed Description

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

polynomial with coefficients in Ring Ring must be an integral domain

Examples

examples/compensated_horner.cpp, examples/make_polynomial.cpp, and examples/modular_arithmetic.cpp.

8.23.2 Member Typedef Documentation

8.23.2.1 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	

8.23.2.2 derive_t

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

derivation operator

Template Parameters



8.23.2.3 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	

8.23.2.4 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	

8.23.2.5 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	

8.23.2.6 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	

8.23.2.7 inject constant t

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

makes the constant (native type) polynomial a_0

Template Parameters



8.23.2.8 inject_ring_t

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

makes the constant (ring type) polynomial a_0

Template Parameters



8.23.2.9 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	

8.23.2.10 mod t

 ${\tt template}{<}{\tt typename~Ring~>}$

```
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	

8.23.2.11 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	

8.23.2.12 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	

8.23.2.13 one

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

constant one

8.23.2.14 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)

Tem	ın	lai	łΔ	Pa	ra	m	ate	a۱	'n
IeII	ıv	ıa	LC.	гα	ıα	1119	= L	C١	э

V	
---	--

8.23.2.15 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



8.23.2.16 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	

8.23.2.17 X

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

generator

8.23.2.18 zero

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

constant zero

8.23.3 Member Data Documentation

8.23.3.1 is euclidean domain

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

8.23.3.2 is field

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

8.23.3.3 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

8.24 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

8.24.1 Detailed Description

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

removes types from head of the list

8.24.2 Member Typedef Documentation

8.24.2.1 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

8.24.2.2 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

8.25 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 > >
        division operator
    template<typename v1 , typename Ring::template div_t< typename v1::type, typename v2::type > >
        division operator
    template<typename v1 , typename v2 >
```

using mod_t = val< typename Ring::template mod_t< typename v1::type, typename v2::type >>

```
    modulus operator
    template<typename v1, typename v2 >
        using eq_t = typename Ring::template eq_t < typename v1::type, typename v2::type >
        equality operator (as type)
    template<typename v1 >
        using pos_t = std::true_type
        positivity operator always true
    template<auto x>
        using inject_constant_t = val < typename Ring::template inject_constant_t < x > >
        inject a 'constant' in quotient ring*
    template<typename v >
        using inject_ring_t = val < v >
        projects a value of Ring onto the quotient
```

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
```

8.25.1 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>

8.25.2 Member Typedef Documentation

8.25.2.1 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

8.25.2.2 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 \leftarrow ::type, typename v2::type> >
```

division operator

Template Parameters

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

8.25.2.3 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

8.25.2.4 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> >
```

inject a 'constant' in quotient ring*

Template Parameters

x a 'constant' from Ring point of view

8.25.2.5 inject_ring_t

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

projects a value of Ring onto the quotient

Template Parameters

```
v a value in Ring
```

8.25.2.6 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 \leftarrow ::type, typename v2::type> >
```

modulus operator

Template Parameters

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

8.25.2.7 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 \leftarrow ::type, typename v2::type> >
```

substraction operator

Template Parameters

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

8.25.2.8 one

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

one

8.25.2.9 pos_t

```
template<typename Ring , typename X >
template<typename v1 >
using aerobus::Quotient< Ring, X >::pos_t = std::true_type
```

positivity operator always true

Template Parameters

```
v1 a value in quotient ring
```

8.25.2.10 zero

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

zero value

8.25.3 Member Data Documentation

8.25.3.1 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

8.25.3.2 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
```

8.25.3.3 pos_v

```
template<typename Ring , typename X >
template<typename v >
constexpr bool aerobus::Quotient< Ring, X >::pos_v = pos_t<v>::value [static], [constexpr]
positivity operator always true
```

Template Parameters

```
v1 a value in quotient ring
```

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

· src/aerobus.h

8.26 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

8.26.1 Detailed Description

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

8.26.2 Member Typedef Documentation

8.26.2.1 head

index

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

8.26.2.2 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

8.27 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
```

8.27.1 Detailed Description

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

Empty pure template struct to handle type list.

A list of types.

Template Parameters

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

8.27.2 Member Typedef Documentation

8.27.2.1 at

```
template<typename... Ts>
template<size_t index>
using aerobus::type_list< Ts >::at = internal::type_at_t<index, Ts...>
```

returns type at index

Template Parameters



8.27.2.2 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



8.27.2.3 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	

8.27.2.4 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
```

8.27.2.5 push_front

Τ

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

Adds T to front of the list.

Template Parameters

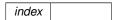


8.27.2.6 remove

```
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



8.27.3 Member Data Documentation

8.27.3.1 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

8.28 aerobus::type_list<> Struct Reference

specialization for empty type list

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

Public Types

```
    template<typename T > using push_front = type_list< T >
    template<typename T > using push_back = type_list< T >
    template<typename U > using concat = U
    template<typename T , size_t index> using insert = type_list< T >
```

Static Public Attributes

• static constexpr size_t length = 0

8.28.1 Detailed Description

specialization for empty type list

8.28.2 Member Typedef Documentation

8.28.2.1 concat

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

8.28.2.2 insert

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

8.28.2.3 push_back

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

8.28.2.4 push_front

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

8.28.3 Member Data Documentation

8.28.3.1 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

8.29 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 DEVICE valueType get ()
        cast x into valueType
    static std::string to_string ()
        string representation of value
```

Static Public Attributes

static constexpr int32_t v = x
 actual value stored in val type

8.29.1 Detailed Description

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

values in i32, again represented as types
```

Template Parameters

```
x an actual integer
```

8.29.2 Member Typedef Documentation

8.29.2.1 enclosing_type

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

Enclosing ring type.

8.29.2.2 is_zero_t

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

is value zero

8.29.3 Member Function Documentation

8.29.3.1 get()

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

cast x into valueType

Template Parameters

```
valueType double for example
```

8.29.3.2 to_string()

8.29.4 Member Data Documentation

8.29.4.1 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

8.30 aerobus::i64::val< x > Struct Template Reference

```
values in i64
#include <aerobus.h>
```

Public Types

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

Static Public Member Functions

```
    template<typename valueType >
    static constexpr INLINED DEVICE valueType get ()
    cast value in valueType
    static std::string to_string ()
    string representation
```

Static Public Attributes

static constexpr int64_t v = x
 actual value

8.30.1 Detailed Description

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

values in i64

Template Parameters
```

```
x an actual integer
```

Examples

examples/compensated_horner.cpp.

8.30.2 Member Typedef Documentation

8.30.2.1 enclosing_type

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

8.30.2.2 inner_type

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

type of represented values

8.30.2.3 is_zero_t

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

is value zero

8.30.3 Member Function Documentation

8.30.3.1 get()

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

cast value in valueType

Template Parameters

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

8.30.3.2 to_string()

string representation

8.30.4 Member Data Documentation

8.30.4.1 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

8.31 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

template<typename x >
 using value_at_t = horner_reduction_t< val > ::template inner< 0, degree+1 > ::template type< typename
 Ring::zero, x >

Static Public Member Functions

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

template < typename arithmeticType >
 static constexpr DEVICE INLINED arithmeticType eval (const arithmeticType &x)
 evaluates polynomial seen as a function operating on arithmeticType

template<typename arithmeticType >
 static DEVICE INLINED arithmeticType compensated_eval (const arithmeticType &x)

Evaluate polynomial on x using compensated horner scheme This is twice as accurate as simple eval (horner) but cannot be constexpr Please not this makes no sense on integer types as arithmetic on integers is exact in IEEE.

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

8.31.1 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

Examples

examples/compensated_horner.cpp.

8.31.2 Member Typedef Documentation

8.31.2.1 aN

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

heavy weight coefficient (non zero)

8.31.2.2 coeff_at_t

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

type of coefficient at index

Template Parameters

index	

8.31.2.3 enclosing_type

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

8.31.2.4 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

8.31.2.5 ring_type

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

ring coefficients live in

8.31.2.6 strip

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

remove largest coefficient

8.31.2.7 value_at_t

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
template<typename x >
using aerobus::polynomial< Ring >::val< coeffN, coeffs >::value_at_t = horner_reduction_t<val>
::template inner<0, degree + 1> ::template type<typename Ring::zero, x>
```

8.31.3 Member Function Documentation

8.31.3.1 compensated_eval()

Evaluate polynomial on x using compensated horner scheme This is twice as accurate as simple eval (horner) but cannot be constexpr Please not this makes no sense on integer types as arithmetic on integers is exact in IEEE.

Template Parameters

Parameters



8.31.3.2 eval()

evaluates polynomial seen as a function operating on arithmeticType

Template Parameters

Parameters

```
x value
```

Returns

P(x)

8.31.3.3 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
```

8.31.4 Member Data Documentation

8.31.4.1 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

8.31.4.2 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

8.32 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 >>

8.32.1 Detailed Description

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

projection values in the quotient ring

```
V a value from 'Ring'
```

98 Class Documentation

8.32.2 Member Typedef Documentation

8.32.2.1 raw_t

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

8.32.2.2 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

8.33 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 INLINED DEVICE valueType get ()
    get value as valueType
    static std::string to_string ()
    string representation
```

Static Public Attributes

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

8.33.1 Detailed Description

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

Template Parameters

```
x an integer
```

8.33.2 Member Typedef Documentation

8.33.2.1 enclosing_type

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

8.33.2.2 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

8.33.3 Member Function Documentation

8.33.3.1 get()

```
template<int32_t p>
template<iint32_t x>
template<typename valueType >
static constexpr INLINED DEVICE valueType aerobus::zpz::val< x >::get ( ) [inline],
[static], [constexpr]
```

get value as valueType

Template Parameters

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

8.33.3.2 to_string()

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

string representation

100 Class Documentation

Returns

a string representation

8.33.4 Member Data Documentation

8.33.4.1 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

8.33.4.2 v

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

actual value

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

• src/aerobus.h

8.34 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
    template < typename x >
        using value_at_t = coeffN
```

Static Public Member Functions

- static std::string to_string ()
- template<typename arithmeticType >
 static constexpr DEVICE INLINED arithmeticType eval (const arithmeticType &x)
- template<typename arithmeticType >
 static DEVICE INLINED arithmeticType compensated_eval (const arithmeticType &x)

Static Public Attributes

- static constexpr size_t degree = 0
 degree
- static constexpr bool is_zero_v = is_zero_t::value

8.34.1 Detailed Description

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

Template Parameters

coeffN

8.34.2 Member Typedef Documentation

8.34.2.1 aN

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

8.34.2.2 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
```

8.34.2.3 enclosing_type

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

enclosing ring type

102 Class Documentation

8.34.2.4 is_zero_t

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

8.34.2.5 ring type

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

ring coefficients live in

8.34.2.6 strip

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

8.34.2.7 value at t

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

8.34.3 Member Function Documentation

8.34.3.1 compensated eval()

8.34.3.2 eval()

8.34.3.3 to_string()

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

8.34.4 Member Data Documentation

8.34.4.1 degree

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

degree

8.34.4.2 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

8.35 aerobus::zpz Struct Template Reference

congruence classes of integers modulo p (32 bits)

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

Classes

struct val

values in zpz

104 Class Documentation

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)
```

8.35.1 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
```

Examples

examples/modular_arithmetic.cpp, and examples/polynomials_over_finite_field.cpp.

8.35.2 Member Typedef Documentation

8.35.2.1 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

8.35.2.2 div_t

 $template < int32_t p >$

106 Class Documentation

```
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

8.35.2.3 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

8.35.2.4 gcd_t

```
template<int32_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

8.35.2.5 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)

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

8.35.2.6 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
```

8.35.2.7 inner_type

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

underlying type for values

8.35.2.8 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

8.35.2.9 mod_t

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

modulo operator

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

108 Class Documentation

8.35.2.10 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

8.35.2.11 one

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

one value

8.35.2.12 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
```

8.35.2.13 sub_t

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

substraction operator

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

8.35.2.14 zero

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

zero value

8.35.3 Member Data Documentation

8.35.3.1 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

8.35.3.2 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

8.35.3.3 is_euclidean_domain

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

always true

8.35.3.4 is_field

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

true iff p is prime

110 Class Documentation

8.35.3.5 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

8.35.3.6 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

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 #ifdef WITH_CUDA_FP16
00016 #include <bit>
00017 #include <cuda_fp16.h>
00018 #endif
00019
00023 #ifdef _MSC_VER
00024 \#define ALIGNED(x) __declspec(align(x))
00025 #define INLINED ___forceinline
00026 #else
00027 #define ALIGNED(x) __attribute__((aligned(x)))
00028 #define INLINED __attribute__((always_inline)) inline
```

```
00029 #endif
00030
00031 #ifdef __CUDACC_
00032 #define DEVICE __host__ __device__
00033 #else
00034 #define DEVICE
00035 #endif
00036
00038
00040
00042
00043 // aligned allocation
00044 namespace aerobus {
00051
          template<typename T>
00052
          T* aligned_malloc(size_t count, size_t alignment) {
00053
              #ifdef _MSC_VER
              return static cast<T*>( aligned malloc(count * sizeof(T), alignment));
00054
00055
              #else
              return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
00057
              #endif
00058
00059 } // namespace aerobus
00060
00061 // concepts
00062 namespace aerobus {
         template <typename R>
00065
          concept IsRing = requires {
00066
              typename R::one;
              typename R::zero;
00067
00068
              typename R::template add_t<typename R::one, typename R::one>;
00069
              typename R::template sub_t<typename R::one, typename R::one>;
00070
              typename R::template mul_t<typename R::one, typename R::one>;
00071
00072
00074
          template <typename R>
00075
          concept IsEuclideanDomain = IsRing<R> && requires {
00076
              typename R::template div_t<typename R::one, typename R::one>;
              typename R::template mod_t<typename R::one, typename R::one>;
00078
              typename R::template gcd_t<typename R::one, typename R::one>;
00079
              typename R::template eq_t<typename R::one, typename R::one>;
00080
              typename R::template pos_t<typename R::one>;
00081
00082
              R::template pos v<typename R::one> == true;
00083
              // typename R::template gt_t<typename R::one, typename R::zero>;
              R::is_euclidean_domain == true;
00084
00085
00086
00088
          template<typename R>
00089
          concept IsField = IsEuclideanDomain<R> && requires {
             R::is_field == true;
00090
00092 } // namespace aerobus
00093
00094 #ifdef WITH_CUDA_FP16
00095 // all this shit is required because of NVIDIA bug https://developer.nvidia.com/bugs/4863696
00096 namespace aerobus {
         namespace internal {
00098
              static consteval DEVICE uint16_t my_internal_float2half(
00099
                 const float f, uint32_t &sign, uint32_t &remainder) {
00100
                  uint32_t x;
                  uint32_t u;
00101
00102
                 uint32 t result;
00103
                  x = std::bit_cast<int32_t>(f);
00104
                  u = (x \& 0x7fffffffU);
00105
                  sign = ((x \gg 16U) \& 0x8000U);
                  // NaN/+Inf/-Inf
00106
00107
                  if (u >= 0x7f800000U) {
00108
                      remainder = 0U:
                      result = ((u == 0x7f800000U) ? (sign | 0x7c00U) : 0x7fffU);
00109
                  } else if (u > 0x477fefffU) { // Overflows
00110
00111
                     remainder = 0x80000000U;
00112
                      result = (sign | 0x7bffU);
                  } else if (u >= 0x38800000U) { // Normal numbers
remainder = u « 19U;
00113
00114
                      u -= 0x38000000U;
00115
00116
                      result = (sign | (u \gg 13U));
00117
                  } else if (u < 0x33000001U) { // +0/-0
00118
                     remainder = u;
                  result = sign;
} else { // Denormal numbers
  const uint32_t exponent = u » 23U;
00119
00120
00121
                      const uint32_t shift = 0x7eU - exponent;
00123
                      uint32_t mantissa = (u & 0x7ffffffU);
00124
                      mantissa |= 0x800000U;
00125
                      remainder = mantissa « (32U - shift);
00126
                      result = (sign | (mantissa » shift));
                      result &= 0x0000FFFFU;
00127
```

```
00129
                   return static_cast<uint16_t>(result);
00130
00131
              static consteval DEVICE __half my_float2half_rn(const float a) {
00132
                 __half val;
__half_raw r;
00133
00134
00135
                   uint32_t sign = 0U;
00136
                  uint32_t remainder = 0U;
00137
                   r.x = my_internal_float2half(a, sign, remainder);
                  if ((remainder > 0x80000000U) || ((remainder == 0x80000000U) && ((r.x & 0x1U) != 0U))) {
00138
00139
                       r.x++;
00140
00141
00142
                  val = std::bit_cast<__half>(r);
00143
                  return val;
00144
              }
00145
00146
              template <int16_t i>
00147
              static constexpr __half convert_int16_to_half = my_float2half_rn(static_cast<float>(i));
00148
00149
00150
              template <typename Out, int16_t x, typename E = void>
00151
              struct int16 convert helper;
00152
00153
              template <typename Out, int16_t x>
00154
              struct int16_convert_helper<Out, x,
00155
                 std::enable_if_t<!std::is_same_v<Out, __half> && !std::is_same_v<Out, __half2>> {
00156
                  static constexpr Out value() {
00157
                       return static_cast<Out>(x);
00158
                  }
00159
              } ;
00160
00161
              template <int16_t x>
              struct int16_convert_helper<__half, x> {
    static constexpr __half value() {
        return convert_int16_to_half<x>;
00162
00163
00164
00165
00166
              };
00167
00168
              template <int16_t x>
              struct int16_convert_helper<__half2, x> {
    static constexpr __half2 value() {
00169
00170
                       return __half2(convert_int16_to_half<x>, convert_int16_to_half<x>);
00171
00172
00173
              } ;
00174
            // namespace internal
00176 #endif
00177
00178 // cast
00179 namespace aerobus {
00180
         namespace internal {
00181
             template<typename Out, typename In>
00182
              struct staticcast {
00183
                 template<auto x>
                  static consteval INLINED DEVICE Out func() {
00185
                       return static_cast<Out>(x);
00186
00187
              };
00188
              #ifdef WITH_CUDA_FP16
00189
00190
              template<>
00191
              struct staticcast<__half, int16_t> {
                  template<int16_t x>
00192
                  static consteval INLINED DEVICE __half func() {
00193
00194
                       return int16_convert_helper<__half, x>::value();
00195
                 }
00196
              };
00197
00198
              template<>
00199
               struct staticcast<__half2, int16_t> {
                 template<int16_t x>
static consteval INLINED DEVICE __half2 func() {
00200
00201
00202
                       return int16 convert helper< half2, x>::value();
00203
00204
              } ;
              #endif
00205
             // namespace internal
00206
00207 } // namespace aerobus
00208
00209 // fma_helper, required because nvidia fails to reconstruct fma for fp16 types
00210 namespace aerobus {
00211
          namespace internal {
00212
              template<typename T>
00213
              struct fma_helper;
00214
```

```
00215
              template<>
00216
              struct fma_helper<double> {
00217
                 static constexpr INLINED DEVICE double eval(const double x, const double y, const double
     z) {
00218
                      return x * v + z;
00219
                }
00220
             } ;
00221
00222
              template<>
00223
              struct fma_helper<float> {
                static constexpr INLINED DEVICE float eval(const float x, const float y, const float z) {
00224
00225
                     return x * y + z;
                }
00226
00227
             };
00228
00229
              template<>
              struct fma_helper<int32_t> {
00230
                 static constexpr INLINED DEVICE int16_t eval(const int16_t x, const int16_t y, const
00231
     int16_t z) {
00232
                      return x * y + z;
00233
00234
             } ;
00235
              template<>
00236
00237
             struct fma_helper<int16_t> {
                 static constexpr INLINED DEVICE int32_t eval(const int32_t x, const int32_t y, const
00238
     int32_t z) {
00239
                      return x * y + z;
00240
00241
             };
00242
00243
              template<>
00244
             struct fma_helper<int64_t> {
00245
                 static constexpr INLINED DEVICE int64_t eval(const int64_t x, const int64_t y, const
     int64_t z) {
00246
                      return x * y + z;
              }
00247
00248
             };
00249
00250
             #ifdef WITH_CUDA_FP16
00251
              template<>
00252
              struct fma_helper<__half> {
                 static constexpr INLINED DEVICE __half eval(const __half x, const __half y, const __half
00253
     z) {
00254
                      #ifdef ___CUDA_ARCH__
00255
                      return __hfma(x, y, z);
00256
                     #else
00257
                     return x * y + z;
00258
                      #endif
00259
                 }
00260
              };
00261
              template<>
00262
              struct fma_helper<__half2> {
00263
__half2 z) {
                 static constexpr INLINED DEVICE __half2 eval(const __half2 x, const __half2 y, const
                      #ifdef ___CUDA_ARCH_
00265
                      return __hfma2(x, y, z);
00266
                      #else
00267
                      return x * y + z;
00268
                      #endif
00269
                 }
00270
             };
00271
              #endif
00272
            // namespace internal
00273 } // namespace aerobus
00274
00275 // compensated horner utilities
00276 namespace aerobus {
00277
         namespace internal {
00278
             template <typename T>
00279
              struct FloatLayout;
00280
00281
              template <>
              struct FloatLayout<double> {
00282
00283
                 static constexpr uint8_t exponent = 11;
00284
                 static constexpr uint8_t mantissa = 53;
00285
                 static constexpr uint8_t r = 27; // ceil(mantissa/2)
00286
              } ;
00287
00288
              template <>
00289
              struct FloatLayout<float> {
00290
                 static constexpr uint8_t exponent = 8;
00291
                 static constexpr uint8_t mantissa = 24;
00292
                  static constexpr uint8_t r = 11; // ceil(mantissa/2)
00293
             };
00294
00295
              #ifdef WITH_CUDA_FP16
```

```
00296
               template <>
               struct FloatLayout<__half> {
00297
00298
                   static constexpr uint8_t exponent = 5;
                   static constexpr uint8_t mantissa = 11; // 10 explicitely stored static constexpr uint8_t r = 6; // ceil(mantissa/2)
00299
00300
00301
               };
00302 #endif
00303
00304
               template<typename T>
               static constexpr INLINED DEVICE void split(T a, T *x, T *y) {
00305
                   T z = a * ((1 « FloatLayout<T>::r) + 1);
*x = z - (z - a);
00306
00307
                   *y = a - *x;
00308
00309
00310
               template<typename T>
00311
               static constexpr INLINED DEVICE void two_sum(T a, T b, T *x, T *y) {
00312
00313
                   *x = a + b;
                   Tz = *x - a;
00314
00315
                   *y = (a - (*x - z)) + (b - z);
00316
00317
00318
               {\tt template}{<}{\tt typename}\ {\tt T}{>}
               static constexpr INLINED DEVICE void two_prod(T a, T b, T *x, T *y) {
00319
00320
                   *x = a * b;
                   T ah, al, bh, bl;
00321
00322
                   split(a, &ah, &al);
                   split(b, &bh, &bl);
*y = al * bl - (((*x - ah * bh) - al * bh) - ah * bl);
00323
00324
00325
               }
00326
00327
00328
               template<typename T, size_t N>
00329
               static INLINED DEVICE T horner(T \starp1, T \starp2, T x) {
                   T r = p1[0] + p2[0];

for (int64_t i = N - 1; i >= 0; --i) {
00330
00331
                       r = r * x + p1[N - i] + p2[N - i];
00332
00333
00334
00335
                   return r;
00336
           } // namespace internal
00337
00338 } // namespace aerobus
00339
00340 // utilities
00341 namespace aerobus {
00342
          namespace internal {
00343
               \label{template} \verb|template| template| typename ...> | typename | TT, | typename | T>
00344
               struct is_instantiation_of : std::false_type { };
00345
00346
               template<template<typename...> typename TT, typename... Ts>
00347
               struct is_instantiation_of<TT, TT<Ts...» : std::true_type { };</pre>
00348
00349
               template<template<typename...> typename TT, typename T>
00350
               inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00351
               template <int64_t i, typename T, typename... Ts>
00353
               struct type_at {
                  static_assert(i < sizeof...(Ts) + 1, "index out of range");
using type = typename type_at<i - 1, Ts...>::type;
00354
00355
00356
               };
00357
00358
               template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00359
                   using type = T;
00360
               } ;
00361
00362
               template <size_t i, typename... Ts>
               using type_at_t = typename type_at<i, Ts...>::type;
00363
00364
00365
00366
               template<size_t n, size_t i, typename E = void>
00367
               struct _is_prime {};
00368
00369
               template<size t i>
00370
               struct _is_prime<0, i> {
                   static constexpr bool value = false;
00371
00372
00373
00374
               template<size_t i>
00375
               struct _is_prime<1, i> {
00376
                   static constexpr bool value = false;
00377
00378
00379
               template<size_t i>
00380
               struct _is_prime<2, i> {
                   static constexpr bool value = true;
00381
00382
```

```
00383
00384
              template<size_t i>
00385
              struct _is_prime<3, i> {
00386
                static constexpr bool value = true;
00387
00388
00389
              template<size_t i>
00390
              struct _is_prime<5, i> {
00391
                static constexpr bool value = true;
00392
00393
00394
              template<size t i>
00395
              struct _is_prime<7, i> {
00396
                  static constexpr bool value = true;
00397
00398
00399
              {\tt template} < {\tt size\_t n, size\_t i} >
              struct_is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)» {
    static constexpr bool value = false;</pre>
00400
00401
00402
              };
00403
00404
              template<size_t n, size_t i>
              00405
00406
                  static constexpr bool value = false;
00407
00408
00409
              template<size_t n, size_t i>
00410
              struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)» {
00411
                 static constexpr bool value = true;
00412
00413
00414
              template<size_t n, size_t i>
00415
              struct _is_prime<n, i, std::enable_if_t<(
00416
                  n % i == 0 &&
                  n >= 9 &&
00417
                  n % 3 != 0 &&
00418
00419
                  n % 2 != 0 &&
                  i * i > n)» {
00421
                  static constexpr bool value = true;
00422
00423
00424
              template<size_t n, size_t i>
              struct _is_prime<n, i, std::enable_if_t<(
    n % (i+2) == 0 &&</pre>
00425
00426
00427
                  n >= 9 &&
00428
                  n % 3 != 0 &&
00429
                  n % 2 != 0 &&
00430
                  i * i <= n) » {
00431
                  static constexpr bool value = true;
00432
00433
00434
              template<size_t n, size_t i>
00435
              struct _is_prime<n, i, std::enable_if_t<(
                      n % (i+2) != 0 &&
n % i != 0 &&
00436
00437
00438
                      n >= 9 &&
00439
                      n % 3 != 0 &&
00440
                      n % 2 != 0 &&
00441
                      (i * i <= n))» {
00442
                  static constexpr bool value = _is_prime<n, i+6>::value;
00443
              };
00444
          } // namespace internal
00445
00448
          template<size_t n>
00449
          struct is_prime {
00451
             static constexpr bool value = internal::_is_prime<n, 5>::value;
00452
00453
00457
          template<size t n>
00458
          static constexpr bool is_prime_v = is_prime<n>::value;
00459
00460
00461
          namespace internal {
00462
              template <std::size_t... Is>
00463
              constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
00464
                  -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
00465
00466
              template <std::size_t N>
00467
              using make_index_sequence_reverse
00468
                  = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00469
00475
              template<typename Ring, typename E = void>
00476
              struct qcd;
00477
00478
              template<typename Ring>
              struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
00479
00480
                  template<typename A, typename B, typename E = void>
```

```
struct gcd_helper {};
00482
00483
                   // B = 0, A > 0
00484
                   template<typename A, typename B>
                  struct gcd_helper<A, B, std::enable_if_t<
    ((B::is_zero_t::value) &&</pre>
00485
00486
                          (Ring::template gt_t<A, typename Ring::zero>::value))» {
00488
                       using type = A;
00489
                  } ;
00490
                  // B = 0, A < 0
00491
                  template<typename A, typename B>
struct gcd_helper<A, B, std::enable_if_t<</pre>
00492
00493
00494
                      ((B::is_zero_t::value) &&
00495
                          !(Ring::template gt_t<A, typename Ring::zero>::value))» {
00496
                       using type = typename Ring::template sub_t<typename Ring::zero, A>;
00497
                  };
00498
00499
                   // B != 0
00500
                   template<typename A, typename B>
00501
                  struct gcd_helper<A, B, std::enable_if_t<
00502
                       (!B::is_zero_t::value)
00503
                       » {
                  private: // NOLINT
00504
00505
                       // A / B
00506
                       using k = typename Ring::template div_t<A, B>;
00507
                       // A - (A/B) *B = A % B
00508
                       using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
00509
00510
                  public:
00511
                      using type = typename gcd_helper<B, m>::type;
00512
00513
00514
                  template<typename A, typename B> \,
00515
                  using type = typename gcd_helper<A, B>::type;
00516
              };
          } // namespace internal
00517
00519
          // vadd and vmul
00520
          namespace internal {
00521
              template<typename... vals>
00522
              struct vmul {};
00523
00524
              template<typename v1, typename... vals>
00525
              struct vmul<v1, vals...> {
00526
                 using type = typename v1::enclosing_type::template mul_t<v1, typename
     vmul<vals...>::type>;
00527
             };
00528
00529
              template<tvpename v1>
00530
              struct vmul<v1> {
00531
                using type = v1;
00532
              };
00533
00534
              template<typename... vals>
00535
              struct vadd {};
00537
              template<typename v1, typename... vals>
00538
              struct vadd<v1, vals...> {
00539
                 using type = typename v1::enclosing_type::template add_t<v1, typename
     vadd<vals...>::type>;
00540
             };
00541
00542
              template<typename v1>
00543
              struct vadd<v1> {
                using type = v1;
00544
00545
              };
          } // namespace internal
00546
00547
          template<typename T, typename A, typename B>
00551
          using gcd_t = typename internal::gcd<T>::template type<A, B>;
00552
00556
          template<typename... vals>
00557
          using vadd_t = typename internal::vadd<vals...>::type;
00558
00562
          template<typename... vals>
00563
          using vmul_t = typename internal::vmul<vals...>::type;
00564
00568
          template < typename val >
00569
          requires IsEuclideanDomain<typename val::enclosing type>
00570
          using abs_t = std::conditional_t<
00571
                           val::enclosing_type::template pos_v<val>,
                           val, typename val::enclosing_type::template
      sub_t<typename val::enclosing_type::zero, val>>;
00573 } // namespace aerobus
00574
00575 // embedding
```

```
00576 namespace aerobus {
00581
        template<typename Small, typename Large, typename E = void>
          struct Embed;
00582
00583 } // namespace aerobus
00584
00585 namespace aerobus {
00590
          template<typename Ring, typename X>
00591
          requires IsRing<Ring>
00592
          struct Quotient {
00595
              \texttt{template} \; \texttt{<typename} \; \texttt{V>} \;
              struct val {
00596
              public:
00597
00598
                  using raw_t = V;
00599
                   using type = abs_t<typename Ring::template mod_t<V, X>>;
00600
              };
00601
              using zero = val<typename Ring::zero>;
00603
00604
00606
              using one = val<typename Ring::one>;
00607
00611
               template<typename v1, typename v2>
00612
              using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00613
00617
              template<typename v1, typename v2>
00618
              using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00619
00623
               template<typename v1, typename v2>
00624
              using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00625
00629
              template<typename v1, typename v2>
00630
              using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00631
00635
              template<typename v1, typename v2>
00636
              using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00637
              template<typename v1, typename v2> \,
00641
              static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00642
00643
00647
               template<typename v1>
00648
              using pos_t = std::true_type;
00649
              template<typename v>
00653
              static constexpr bool pos_v = pos_t<v>::value;
00654
00655
00657
              static constexpr bool is_euclidean_domain = true;
00658
00662
              template<auto x>
00663
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00664
00668
              template<tvpename v>
00669
              using inject_ring_t = val<v>;
00670
00671
00675
          template<typename Ring, typename X>
00676
          struct Embed<Quotient<Ring, X>, Ring> {
00679
              template<typename val>
00680
              using type = typename val::raw_t;
00681
00682 }
        // namespace aerobus
00683
00684 // type list
00685 namespace aerobus {
00687
          template <typename... Ts>
00688
          struct type_list;
00689
00690
          namespace internal {
00691
              template <typename T, typename... Us>
00692
               struct pop_front_h {
                  using tail = type_list<Us...>;
00693
                  using head = T;
00694
00695
              };
00696
00697
              template <size_t index, typename L1, typename L2>
              struct split_h {
00698
00699
               private:
00700
                  static_assert(index <= L2::length, "index ouf of bounds");</pre>
00701
                  using a = typename L2::pop_front::type;
00702
                  using b = typename L2::pop_front::tail;
00703
                  using c = typename L1::template push_back<a>;
00704
00705
                public:
                  using head = typename split_h<index - 1, c, b>::head; using tail = typename split_h<index - 1, c, b>::tail;
00706
00707
00708
00709
00710
              template <typename L1, typename L2>
struct split_h<0, L1, L2> {
00711
```

```
00712
                   using head = L1;
                   using tail = L2;
00713
00714
               };
00715
00716
               template <size_t index, typename L, typename T>
00717
               struct insert h {
00718
                   static_assert(index <= L::length, "index ouf of bounds");</pre>
00719
                   using s = typename L::template split<index>;
                   using left = typename s::head;
using right = typename s::tail;
00720
00721
                   using 11 = typename left::template push_back<T>;
00722
00723
                   using type = typename ll::template concat<right>;
00724
               };
00725
00726
               template <size_t index, typename L>
00727
               struct remove_h {
                   using s = typename L::template split<index>;
using left = typename s::head;
using right = typename s::tail;
00728
00729
00730
00731
                   using rr = typename right::pop_front::tail;
00732
                   using type = typename left::template concat<rr>;
00733
          } // namespace internal
00734
00735
00738
          template <typename... Ts>
00739
          struct type_list {
00740
           private:
00741
               template <typename T>
00742
               struct concat_h;
00743
00744
               template <typename... Us>
00745
               struct concat_h<type_list<Us...» {
00746
                  using type = type_list<Ts..., Us...>;
00747
               };
00748
00749
           public:
00751
               static constexpr size t length = sizeof...(Ts);
00752
00755
               template <typename T>
00756
               using push_front = type_list<T, Ts...>;
00757
               template <size_t index>
using at = internal::type_at_t<index, Ts...>;
00760
00761
00762
00764
               struct pop_front {
00766
                   using type = typename internal::pop_front_h<Ts...>::head;
00768
                   using tail = typename internal::pop_front_h<Ts...>::tail;
00769
               };
00770
00773
               template <typename T>
00774
               using push_back = type_list<Ts..., T>;
00775
00778
               template <typename U>
00779
               using concat = typename concat_h<U>::type;
00780
00783
               template <size t index>
00784
               struct split {
00785
               private:
00786
                   using inner = internal::split_h<index, type_list<>, type_list<Ts...»;</pre>
00787
00788
                public:
00789
                  using head = typename inner::head;
00790
                   using tail = typename inner::tail;
00791
00792
00796
               template <typename T, size_t index>
00797
               using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00798
00801
               template <size_t index>
00802
               using remove = typename internal::remove_h<index, type_list<Ts...»::type;</pre>
00803
00804
00806
          template <>
          struct type_list<> {
00807
00808
               static constexpr size t length = 0;
00809
00810
               template <typename T>
00811
               using push_front = type_list<T>;
00812
00813
               template <typename T>
00814
               using push_back = type_list<T>;
00815
00816
               template <typename U>
00817
               using concat = U;
00818
               // TODO(jewave): assert index == 0
00819
00820
               template <typename T, size_t index>
```

```
using insert = type_list<T>;
00822
00823 } // namespace aerobus
00824
00825 // i16
00826 #ifdef WITH_CUDA_FP16
00827 // i16
00828 namespace aerobus {
00830
         struct i16 {
              using inner_type = int16_t;
00831
00834
              {\tt template}{<} {\tt int16\_t} \ x{>}
00835
              struct val {
00837
                  using enclosing_type = i16;
                  static constexpr int16_t v = x;
00839
00840
00843
                  template<typename valueType>
                  static constexpr INLINED DEVICE valueType get() {
00844
00845
                      return internal::template int16_convert_helper<valueType, x>::value();
00846
00847
00849
                  using is_zero_t = std::bool_constant<x == 0>;
00850
00852
                  static std::string to_string() {
00853
                      return std::to_string(x);
00854
                  }
00855
              };
00856
00858
              using zero = val<0>;
              using one = val<1>;
00860
              static constexpr bool is_field = false;
00862
00864
              static constexpr bool is_euclidean_domain = true;
00867
              template<auto x>
00868
              using inject_constant_t = val<static_cast<int16_t>(x)>;
00869
00870
              {\tt template}{<}{\tt typename}\ {\tt v}{>}
00871
              using inject_ring_t = v;
00872
00874
              template<typename v1, typename v2>
00875
              struct add {
00876
                  using type = val<v1::v + v2::v>;
00877
              };
00878
00879
              template<typename v1, typename v2>
00880
              struct sub {
00881
                  using type = val<v1::v - v2::v>;
00882
00883
00884
              template<typename v1, typename v2>
00885
              struct mul {
00886
                  using type = val<v1::v* v2::v>;
00887
00888
00889
              template<typename v1, typename v2>
00890
              struct div {
00891
                  using type = val<v1::v / v2::v>;
00892
00893
00894
              template<typename v1, typename v2>
              struct remainder {
    using type = val<v1::v % v2::v>;
00895
00896
00897
00898
00899
              template<typename v1, typename v2>
00900
00901
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00902
00903
00904
              template<tvpename v1, tvpename v2>
00905
              struct lt {
00906
                 using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00907
00908
00909
              template<typename v1, typename v2>
00910
              struct eq {
00911
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00912
00913
00914
              template<typename v1>
00915
              struct pos {
00916
                  using type = std::bool_constant<(v1::v > 0)>;
00917
00918
00919
           public:
00924
              template<typename v1, typename v2> \,
00925
              using add_t = typename add<v1, v2>::type;
00926
```

```
template<typename v1, typename v2>
00932
              using sub_t = typename sub<v1, v2>::type;
00933
00938
              template<typename v1, typename v2>
00939
              using mul t = typename mul<v1, v2>::type;
00940
00945
              template<typename v1, typename v2>
00946
              using div_t = typename div<v1, v2>::type;
00947
00952
              template<typename v1, typename v2>
00953
              using mod_t = typename remainder<v1, v2>::type;
00954
              template<typename v1, typename v2>
00959
00960
              using gt_t = typename gt<v1, v2>::type;
00961
00966
              template<typename v1, typename v2>
00967
              using lt_t = typename lt<v1, v2>::type;
00968
              template<typename v1, typename v2>
00974
              using eq_t = typename eq<v1, v2>::type;
00975
00979
              template<typename v1, typename v2>
00980
              static constexpr bool eq_v = eq_t<v1, v2>::value;
00981
00986
              template<typename v1, typename v2>
              using gcd_t = gcd_t<i16, v1, v2>;
00987
00988
00992
              template<typename v>
00993
              using pos_t = typename pos<v>::type;
00994
00998
              template<tvpename v>
00999
              static constexpr bool pos_v = pos_t<v>::value;
01000
01001 } // namespace aerobus
01002 #endif
01003
01004 // i32
01005 namespace aerobus {
01007
         struct i32 {
01008
            using inner_type = int32_t;
01011
              template<int32_t x>
01012
              struct val {
                 using enclosing_type = i32;
01014
01016
                  static constexpr int32_t v = x;
01017
01020
                  template<typename valueType>
01021
                  static constexpr DEVICE valueType get() {
01022
                      return static_cast<valueType>(x);
                  }
01023
01024
01026
                  using is_zero_t = std::bool_constant<x == 0>;
01027
01029
                  static std::string to_string() {
01030
                     return std::to_string(x);
                  }
01031
01032
              };
01035
              using zero = val<0>;
01037
              using one = val<1>;
01039
              static constexpr bool is_field = false;
01041
              static constexpr bool is_euclidean_domain = true;
01044
              template<auto x>
01045
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01046
01047
              template<typename v>
01048
              using inject_ring_t = v;
01049
01050
           private:
             template<typename v1, typename v2>
01051
              struct add {
01053
                 using type = val<v1::v + v2::v>;
01054
01055
01056
              template<typename v1, typename v2> ^{\circ}
01057
              struct sub {
01058
                  using type = val<v1::v - v2::v>;
01059
01060
01061
              template<typename v1, typename v2>
01062
              struct mul {
                 using type = val<v1::v* v2::v>;
01063
01064
01065
01066
              template<typename v1, typename v2>
              struct div {
01067
                  using type = val<v1::v / v2::v>;
01068
01069
              };
```

```
01070
01071
              template<typename v1, typename v2>
01072
              struct remainder {
                using type = val<v1::v % v2::v>;
01073
01074
01075
01076
              template<typename v1, typename v2>
01077
01078
                using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01079
01080
01081
              template<typename v1, typename v2>
01082
              struct lt {
01083
                 using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01084
01085
01086
              template<typename v1, typename v2>
01087
              struct eq {
01088
                 using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01089
01090
01091
              template<typename v1>
01092
              struct pos {
                  using type = std::bool_constant<(v1::v > 0)>;
01093
01094
              };
01095
           public:
01096
01101
              template<typename v1, typename v2>
01102
              using add_t = typename add<v1, v2>::type;
01103
              template<typename v1, typename v2>
01108
01109
              using sub_t = typename sub<v1, v2>::type;
01110
01115
              template<typename v1, typename v2>
01116
              using mul_t = typename mul<v1, v2>::type;
01117
              template<typename v1, typename v2>
01122
01123
              using div_t = typename div<v1, v2>::type;
01124
01129
              template<typename v1, typename v2>
01130
              using mod_t = typename remainder<v1, v2>::type;
01131
01136
              template<typename v1, typename v2>
              using gt_t = typename gt<v1, v2>::type;
01137
01138
01143
              template<typename v1, typename v2>
01144
              using lt_t = typename lt<v1, v2>::type;
01145
01150
              template<typename v1, typename v2>
01151
              using eq_t = typename eq<v1, v2>::type;
01152
01156
              template<typename v1, typename v2>
01157
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01158
              template<typename v1, typename v2>
01163
01164
             using gcd t = gcd t < i32, v1, v2>;
01165
01169
              template<typename v>
01170
              using pos_t = typename pos<v>::type;
01171
              template<typename v>
01175
01176
             static constexpr bool pos_v = pos_t<v>::value;
01177
          };
01178 } // namespace aerobus
01179
01180 // i64
01181 namespace aerobus {
01183
         struct i64 {
01185
             using inner_type = int64_t;
              template<int64_t x>
01188
01189
              struct val {
                  using inner_type = int32_t;
01191
01193
                 using enclosing_type = i64;
                 static constexpr int64_t v = x;
01195
01196
01199
                  template<typename valueType>
01200
                  static constexpr INLINED DEVICE valueType get() {
01201
                     return static_cast<valueType>(x);
01202
                  }
01203
                  using is_zero_t = std::bool_constant<x == 0>;
01205
01206
01208
                  static std::string to_string() {
01209
                      return std::to_string(x);
01210
              };
01211
01212
```

```
01215
              template<auto x>
01216
              using inject_constant_t = val<static_cast<int64_t>(x)>;
01217
01222
              template<typename v>
              using inject_ring_t = v;
01223
01224
01226
              using zero = val<0>;
01228
              using one = val<1>;
01230
              static constexpr bool is_field = false;
01232
              static constexpr bool is_euclidean_domain = true;
01233
01234
           private:
01235
              template<typename v1, typename v2>
01236
              struct add {
01237
                 using type = val<v1::v + v2::v>;
01238
01239
01240
              template<typename v1, typename v2>
              struct sub {
01242
                 using type = val<v1::v - v2::v>;
01243
01244
01245
              template<typename v1, typename v2>
01246
              struct mul {
01247
                  using type = val<v1::v* v2::v>;
01248
01249
01250
              template<typename v1, typename v2>
01251
              struct div {
01252
                  using type = val<v1::v / v2::v>;
01253
01254
01255
              template<typename v1, typename v2>
01256
              struct remainder {
01257
                  using type = val<v1::v% v2::v>;
01258
01259
01260
              template<typename v1, typename v2>
01261
              struct gt {
01262
                 using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01263
01264
01265
              template<typename v1, typename v2>
01266
              struct lt {
01267
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01268
01269
01270
              template<typename v1, typename v2>
01271
              struct eq {
                 using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01272
01274
01275
              template<typename v>
01276
              struct pos {
01277
                  using type = std::bool_constant<(v::v > 0)>;
01278
              };
01279
01280
01284
              template<typename v1, typename v2>
01285
              using add_t = typename add<v1, v2>::type;
01286
01290
              template<typename v1, typename v2>
01291
              using sub_t = typename sub<v1, v2>::type;
01292
01296
              template<typename v1, typename v2>
01297
              using mul_t = typename mul<v1, v2>::type;
01298
01303
              template<typename v1, typename v2>
01304
              using div t = typename div<v1, v2>::type;
01305
01309
              template<typename v1, typename v2>
01310
              using mod_t = typename remainder<v1, v2>::type;
01311
01316
              template<typename v1, typename v2>
01317
              using gt_t = typename gt<v1, v2>::type;
01318
01323
              template<typename v1, typename v2>
01324
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01325
01330
              template<typename v1, typename v2> \,
01331
              using lt_t = typename lt<v1, v2>::type;
01332
              template<typename v1, typename v2>
static constexpr bool lt_v = lt_t<v1, v2>::value;
01337
01338
01339
01344
              template<typename v1, typename v2>
01345
              using eq_t = typename eq<v1, v2>::type;
```

```
01346
01351
              template<typename v1, typename v2>
01352
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01353
01358
              template<typename v1, typename v2>
              using gcd_t = gcd_t<i64, v1, v2>;
01359
01360
01364
              template<typename v>
01365
              using pos_t = typename pos<v>::type;
01366
01370
              template<typename v>
01371
              static constexpr bool pos_v = pos_t<v>::value;
01372
          };
01373
01375
          template<>
01376
          struct Embed<i32, i64> {
             template<typename val>
using type = i64::val<static_cast<int64_t>(val::v)>;
01379
01380
01381
01382 } // namespace aerobus
01383
01384 // z/pz
01385 namespace aerobus {
          template<int32_t p>
01391
01392
          struct zpz {
01394
             using inner_type = int32_t;
01395
01398
              template < int32_t x >
01399
              struct val {
                  using enclosing_type = zpz;
01401
                  static constexpr int32_t v = x % p;
01403
01404
01407
                  template<typename valueType>
01408
                  static constexpr INLINED DEVICE valueType get() {
01409
                      return static_cast<valueType>(x % p);
01410
01411
01413
                  using is_zero_t = std::bool_constant<v == 0>;
01414
01416
                  static constexpr bool is_zero_v = v == 0;
01417
01420
                  static std::string to_string() {
01421
                      return std::to_string(x % p);
01422
                  }
01423
              };
01424
01427
              template<auto x>
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01428
01429
01431
              using zero = val<0>:
01432
01434
              using one = val<1>;
01435
01437
              static constexpr bool is_field = is_prime::value;
01438
              static constexpr bool is_euclidean_domain = true;
01440
01442
01443
              template<typename v1, typename v2>
01444
              struct add {
                  using type = val<(v1::v + v2::v) % p>;
01445
01446
01447
01448
              template<typename v1, typename v2>
01449
              struct sub {
01450
                  using type = val<(v1::v - v2::v) % p>;
01451
01452
              template<typename v1, typename v2>
01453
01454
              struct mul {
01455
                 using type = val<(v1::v* v2::v) % p>;
01456
01457
              template<typename v1, typename v2> ^{\circ}
01458
01459
              struct div {
01460
                  using type = val<(v1::v% p) / (v2::v % p)>;
01461
01462
01463
              template<typename v1, typename v2>
01464
              struct remainder {
                  using type = val<(v1::v% v2::v) % p>;
01465
01466
01467
01468
              template<typename v1, typename v2>
01469
              struct gt {
                  using type = std::conditional_t < (v1::v% p > v2::v% p), std::true\_type, std::false\_type>;
01470
01471
              };
```

```
01472
              template<typename v1, typename v2>
01473
              struct lt {
01474
                 using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01475
01476
01477
01478
              template<typename v1, typename v2>
01479
              struct eq {
01480
                using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01481
01482
01483
              template<typename v1>
01484
              struct pos {
01485
                  using type = std::bool_constant<(v1::v > 0)>;
01486
01487
01488
           public:
01492
              template<typename v1, typename v2>
01493
              using add_t = typename add<v1, v2>::type;
01494
01498
              template<typename v1, typename v2>
01499
              using sub_t = typename sub<v1, v2>::type;
01500
01504
              template<typename v1, typename v2>
01505
              using mul_t = typename mul<v1, v2>::type;
01506
01510
              template<typename v1, typename v2>
01511
              using div_t = typename div<v1, v2>::type;
01512
01516
              template<typename v1, typename v2>
01517
              using mod t = typename remainder<v1, v2>::type;
01518
01522
              template<typename v1, typename v2>
01523
              using gt_t = typename gt<v1, v2>::type;
01524
01528
              template<typename v1, typename v2>
01529
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01534
              template<typename v1, typename v2>
01535
              using lt_t = typename lt<v1, v2>::type;
01536
              template<typename v1, typename v2>
static constexpr bool lt_v = lt_t<v1, v2>::value;
01540
01541
01542
01546
              template<typename v1, typename v2>
01547
                          = typename eq<v1, v2>::type;
01548
01552
              template<typename v1, typename v2> ^{\circ}
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01553
01554
              template<typename v1, typename v2>
01559
              using gcd_t = gcd_t<i32, v1, v2>;
01560
01563
              template<typename v1>
01564
              using pos_t = typename pos<v1>::type;
01565
              template<typename v>
01569
              static constexpr bool pos_v = pos_t<v>::value;
01570
         };
01571
          template<int32_t x>
01574
01575
          struct Embed<zpz<x>, i32> {
              template <typename val>
01579
              using type = i32::val<val::v>;
01580
01581 } // namespace aerobus
01582
01583 // polynomial
01584 namespace aerobus {
         // coeffN x^N + ...
01590
          template<typename Ring>
01591
          requires IsEuclideanDomain<Ring>
01592
          struct polynomial {
              static constexpr bool is field = false;
01593
              static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01594
01595
01598
              template<typename P>
01599
              struct horner_reduction_t {
01600
                  template<size_t index, size_t stop>
01601
                  struct inner {
01602
                      template<typename accum, typename x>
01603
                      using type = typename horner_reduction_t<P>::template inner<index + 1, stop>
                           ::template type<
01604
01605
                               typename Ring::template add_t<
01606
                                   typename Ring::template mul_t<x, accum>,
01607
                                   typename P::template coeff_at_t<P::degree - index>
01608
                               >, x>;
```

```
};
01610
01611
                  template<size_t stop>
01612
                  struct inner<stop, stop> {
01613
                      template<typename accum, typename x>
01614
                      using type = accum;
01615
                  };
01616
              };
01617
01621
              template<typename coeffN, typename... coeffs>
01622
              struct val {
01624
                 using ring_type = Ring;
01626
                  using enclosing_type = polynomial<Ring>;
01628
                  static constexpr size_t degree = sizeof...(coeffs);
01630
                  using aN = coeffN;
01632
                  using strip = val<coeffs...>;
                  using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01634
                  static constexpr bool is_zero_v = is_zero_t::value;
01636
01637
01638
               private:
01639
                  template<size_t index, typename E = void>
01640
                  struct coeff_at {};
01641
                  template<size_t index>
01642
01643
                  struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))» {</pre>
                     using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01644
01645
01646
01647
                  template<size_t index>
01648
                  struct coeff at<index, std::enable if t<(index < 0 || index > sizeof...(coeffs))» {
01649
                      using type = typename Ring::zero;
01650
01651
01652
               public:
01655
                  template<size_t index>
                  using coeff_at_t = typename coeff_at<index>::type;
01656
01657
01660
                  static std::string to_string() {
01661
                       return string_helper<coeffN, coeffs...>::func();
01662
01663
01668
                  template<typename arithmeticType>
                  static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01669
                       #ifdef WITH_CUDA_FP16
01670
01671
                       arithmeticType start;
01672
                       if constexpr (std::is_same_v<arithmeticType, __half2>) {
01673
                          start = \underline{\underline{\hspace{0.5cm}}}half2(0, 0);
01674
                       } else {
01675
                          start = static cast<arithmeticTvpe>(0);
01676
01677
01678
                       arithmeticType start = static_cast<arithmeticType>(0);
01679
                       #endif
01680
                       return horner evaluation<arithmeticType, val>
01681
                               ::template inner<0, degree + 1>
01682
                               ::func(start, x);
01683
                  }
01684
01691
                  template<typename arithmeticType>
01692
                  static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01693
                       return compensated horner<arithmeticType, val>::func(x);
01694
                  }
01695
01696
                  template<typename x>
01697
                  using value_at_t = horner_reduction_t<val>
01698
                       ::template inner<0, degree + 1>
01699
                       ::template type<typename Ring::zero, x>;
01700
              };
01701
01704
              template<typename coeffN>
              struct val<coeffN> {
   using ring_type = Ring;
01705
01707
                  using enclosing_type = polynomial<Ring>;
01709
01711
                  static constexpr size_t degree = 0;
01712
                  using aN = coeffN;
01713
                  using strip = val<coeffN>;
01714
                  using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01715
01716
                  static constexpr bool is_zero_v = is_zero_t::value;
01717
01718
                  template<size t index, typename E = void>
                  struct coeff_at {};
01720
01721
                  template<size_t index>
01722
                  struct coeff_at<index, std::enable_if_t<(index == 0)» {</pre>
01723
                       using type = aN;
01724
                  };
```

```
01725
01726
                  template<size_t index>
01727
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01728
                      using type = typename Ring::zero;
01729
01730
01731
                  template<size_t index>
01732
                  using coeff_at_t = typename coeff_at<index>::type;
01733
01734
                  static std::string to_string() {
01735
                      return string_helper<coeffN>::func();
01736
01737
01738
                  template<typename arithmeticType>
01739
                  static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01740
                      return coeffN::template get<arithmeticType>();
01741
                  }
01742
01743
                  template<typename arithmeticType>
                  static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01744
01745
                      return coeffN::template get<arithmeticType>();
01746
01747
01748
                  template<typename x>
01749
                  using value_at_t = coeffN;
01750
              };
01751
01753
              using zero = val<typename Ring::zero>;
01755
              using one = val<typename Ring::one>;
              using X = val<typename Ring::one, typename Ring::zero>;
01757
01758
01759
           private:
01760
              template<typename P, typename E = void>
01761
              struct simplify;
01762
01763
              template <typename P1, typename P2, typename I>
01764
              struct add_low;
01765
01766
              template<typename P1, typename P2>
01767
              struct add {
01768
                  using type = typename simplify<typename add_low<
01769
                  P1,
01770
                  P2.
01771
                  internal::make_index_sequence_reverse<</pre>
01772
                  std::max(P1::degree, P2::degree) + 1
01773
                  »::type>::type;
01774
01775
01776
              template <typename P1, typename P2, typename I>
01777
              struct sub low:
01778
01779
              template <typename P1, typename P2, typename I>
01780
              struct mul_low;
01781
01782
              template<typename v1, typename v2>
01783
              struct mul {
01784
                      using type = typename mul_low<
01785
01786
                          v2,
01787
                           internal::make_index_sequence_reverse<
                          v1::degree + v2::degree + 1
01788
01789
                          »::type;
01790
              };
01791
01792
              template<typename coeff, size_t deg>
01793
              struct monomial;
01794
01795
              template<typename v, typename E = void>
01796
              struct derive helper {};
01797
01798
              template<typename v>
01799
              struct derive_helper<v, std::enable_if_t<v::degree == 0» {</pre>
01800
                  using type = zero;
01801
01802
01803
              template<typename v>
01804
              struct derive_helper<v, std::enable_if_t<v::degree != 0» {
01805
                  using type = typename add<
01806
                       typename derive_helper<typename simplify<typename v::strip>::type>::type,
01807
                      typename monomial<
                          typename Ring::template mul_t<
    typename v::aN,</pre>
01808
01809
01810
                               typename Ring::template inject_constant_t<(v::degree)>
01811
01812
                          v::degree - 1
01813
                      >::type
01814
                  >::type;
```

```
01815
01816
01817
               template<typename v1, typename v2, typename E = void>
01818
               struct eq_helper {};
01819
               template<typename v1, typename v2>
01820
              struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
01821
01822
                   using type = std::false_type;
01823
01824
01825
              template<typename v1, typename v2>
struct eq_helper<v1, v2, std::enable_if_t<
    v1::degree == v2::degree &&</pre>
01826
01827
01828
01829
                   (v1::degree != 0 || v2::degree != 0) &&
01830
                   std::is_same<
01831
                   typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
                   std::false_type
01832
01833
                   >::value
01834
               > {
01835
01836
                   using type = std::false_type;
01837
               };
01838
01839
               template<typename v1, typename v2>
               struct eq_helper<v1, v2, std::enable_if_t<
01840
01841
                   v1::degree == v2::degree &&
01842
                   (v1::degree != 0 || v2::degree != 0) &&
01843
                   std::is_same<
01844
                   typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01845
                   std::true_type
01846
                   >::value
01847
01848
                   using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01849
01850
01851
               template<typename v1, typename v2>
               struct eq_helper<v1, v2, std::enable_if_t<
01852
01853
                   v1::degree == v2::degree &&
01854
                   (v1::degree == 0)
01855
               » {
01856
                   using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01857
               }:
01858
01859
               template<typename v1, typename v2, typename E = void>
01860
               struct lt_helper {};
01861
              template<typename v1, typename v2>
struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
    using type = std::true_type;</pre>
01862
01863
01864
01865
               };
01866
01867
               template<typename v1, typename v2>
01868
               struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
                   using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01869
01870
               };
01871
01872
               template<typename v1, typename v2>
01873
               struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
01874
                   using type = std::false_type;
01875
01876
01877
               template<typename v1, typename v2, typename E = void>
01878
               struct gt_helper {};
01879
               01880
01881
                  using type = std::true_type;
01882
01883
               template<typename v1, typename v2>
struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01885
01886
01887
                   using type = std::false_type;
01888
01889
               template<typename v1, typename v2>
01890
01891
               struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {</pre>
01892
                  using type = std::false_type;
01893
               };
01894
               // when high power is zero : strip
01895
01896
               template<typename P>
01897
               struct simplify<P, std::enable_if_t<
01898
                   std::is_same<
01899
                   typename Ring::zero,
                  typename P::aN
>::value && (P::degree > 0)
01900
01901
```

```
» {
01903
                   using type = typename simplify<typename P::strip>::type;
01904
              };
01905
              // otherwise : do nothing
01906
01907
               template<tvpename P>
              struct simplify<P, std::enable_if_t<
01908
01909
                   !std::is_same<
01910
                  typename Ring::zero,
01911
                  typename P::aN
                  >::value && (P::degree > 0)
01912
01913
              » {
01914
                  using type = P;
01915
              };
01916
01917
              \ensuremath{//} do not simplify constants
01918
              template<tvpename P>
01919
              struct simplify<P, std::enable_if_t<P::degree == 0» {</pre>
                  using type = P;
01920
01921
              };
01922
01923
              // addition at
01924
              template<typename P1, typename P2, size_t index>
01925
              struct add at {
01926
                  using type =
01927
                      typename Ring::template add_t<
01928
                           typename P1::template coeff_at_t<index>,
01929
                           typename P2::template coeff_at_t<index»;</pre>
01930
              };
01931
01932
               template<typename P1, typename P2, size_t index>
01933
              using add_at_t = typename add_at<P1, P2, index>::type;
01934
01935
               template<typename P1, typename P2, std::size_t... I>
01936
              struct add_low<P1, P2, std::index_sequence<I...» {</pre>
01937
                  using type = val<add_at_t<P1, P2, I>...>;
01938
              };
01939
01940
               // substraction at
01941
               template<typename P1, typename P2, size_t index>
01942
               struct sub_at {
01943
                  using type =
01944
                      typename Ring::template sub_t<
01945
                           typename P1::template coeff_at_t<index>,
01946
                           typename P2::template coeff_at_t<index»;</pre>
01947
01948
01949
               template<typename P1, typename P2, size_t index>
              using sub_at_t = typename sub_at<P1, P2, index>::type;
01950
01951
01952
               template<typename P1, typename P2, std::size_t... I>
01953
              struct sub_low<P1, P2, std::index_sequence<I...» {
01954
                  using type = val<sub_at_t<P1, P2, I>...>;
01955
01956
              template<typename P1, typename P2>
01957
01958
               struct sub {
01959
                   using type = typename simplify<typename sub_low<
01960
                  P1,
                  P2.
01961
01962
                  internal::make_index_sequence_reverse<</pre>
01963
                   std::max(P1::degree, P2::degree) + 1
01964
                   »::type>::type;
01965
01966
01967
               // multiplication at
01968
               template<typename v1, typename v2, size_t k, size_t index, size_t stop>
              struct mul_at_loop_helper {
01969
                   using type = typename Ring::template add_t<
01970
01971
                       typename Ring::template mul_t<</pre>
01972
                       typename v1::template coeff_at_t<index>,
01973
                       typename v2::template coeff_at_t<k - index>
01974
01975
                       typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
01976
                   >;
01977
              };
01978
01979
               template<typename v1, typename v2, size_t k, size_t stop>
01980
               struct mul_at_loop_helper<v1, v2, k, stop, stop> {
                  using type = typename Ring::template mul_t
typename v1::template coeff_at_t<stop>,
01981
01982
                       typename v2::template coeff_at_t<0»;
01983
01984
01985
01986
               template <typename v1, typename v2, size_t k, typename E = void>
01987
              struct mul at {};
01988
```

```
template<typename v1, typename v2, size_t k>
01990
               struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) \mid \mid (k > v1::degree + v2::degree)» {
01991
                  using type = typename Ring::zero;
01992
01993
01994
              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)» {
01995
01996
                  using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01997
01998
01999
              template<typename P1, typename P2, size_t index>
02000
              using mul_at_t = typename mul_at<P1, P2, index>::type;
02001
02002
               template<typename P1, typename P2, std::size_t... I>
02003
               struct mul_low<P1, P2, std::index_sequence<I...» {
02004
                  using type = val<mul_at_t<P1, P2, I>...>;
02005
              };
02006
              // division helper
02007
02008
              template< typename A, typename B, typename Q, typename R, typename E = void>
02009
              struct div_helper {};
02010
02011
              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>
02012
02013
                   (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
02014
                   using q_type = Q;
02015
02016
                  using mod_type = R;
02017
                  using gcd_type = B;
02018
              };
02019
02020
               template<typename A, typename B, typename Q, typename R>
02021
              struct div_helper<A, B, Q, R, std::enable_if_t<
02022
                   (R::degree >= B::degree) &&
02023
                   !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
               private: // NOLINT
02024
                  using rN = typename R::aN;
02025
                  using bN = typename B::aN;
02027
                   using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
     B::degree>::type;
02028
                  using rr = typename sub<R, typename mul<pT, B>::type>::type;
02029
                  using qq = typename add<Q, pT>::type;
02030
02031
               public:
02032
                  using q_type = typename div_helper<A, B, qq, rr>::q_type;
02033
                   using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
02034
                  using gcd_type = rr;
02035
              };
02036
02037
              template<typename A, typename B>
              struct div {
02038
02039
                  static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
                  using q_type = typename div_helper<A, B, zero, A>::q_type; using m_type = typename div_helper<A, B, zero, A>::mod_type;
02040
02041
02042
              };
02043
02044
              template<typename P>
02045
02046
                  using type = typename div<P, val<typename P::aN»::q_type;
02047
02048
              template<typename coeff, size_t deg>
02049
02050
              struct monomial {
02051
                  using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
02052
              } ;
02053
02054
              template<typename coeff>
02055
              struct monomial < coeff. 0>
02056
                  using type = val<coeff>;
02057
02058
02059
              template<typename arithmeticType, typename P>
02060
              struct horner_evaluation {
02061
                  template<size_t index, size_t stop>
02062
                  struct inner {
02063
                       static constexpr DEVICE INLINED arithmeticType func(
02064
                          const arithmeticType& accum, const arithmeticType& x) {
02065
                           return horner_evaluation<arithmeticType, P>::template inner<index + 1,</pre>
      stop>::func(
02066
                               internal::fma helper<arithmeticType>::eval(
02067
                                   х,
02068
                                    accum,
                                    P::template coeff_at_t<P::degree - index>::template
      get<arithmeticType>()), x);
02070
02071
                  };
02072
```

```
template<size_t stop>
02074
                    struct inner<stop, stop> {
                        static constexpr DEVICE INLINED arithmeticType func(
02075
02076
                            const arithmeticType& accum, const arithmeticType& x) {
02077
                             return accum;
02078
                        }
02079
                   };
02080
02081
02082
               template<typename arithmeticType, typename P>
               struct compensated_horner {
02083
02084
                  template<int64 t index, int ghost>
02085
                   struct EFTHorner {
02086
                       static INLINED void func(
02087
                                 \verb|arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType | \\
      *r) {
02088
                            arithmeticType p;
                            internal::two_prod(*r, x, &p, pi + P::degree - index - 1);
constexpr arithmeticType coeff = P::template coeff_at_t<index>::template
02089
02090
      get<arithmeticType>();
02091
                             internal::two_sum<arithmeticType>(
                                p, coeff,
r, sigma + P::degree - index - 1);
02092
02093
02094
                            EFTHorner<index - 1, ghost>::func(x, pi, sigma, r);
02095
02096
                   };
02097
02098
                    template<int ghost>
                   struct EFTHorner<-1, ghost> {
    static INLINED DEVICE void func(
02099
02100
                                 arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType
02101
      *r) {
02102
02103
                   };
02104
                    static INLINED DEVICE arithmeticType func(arithmeticType x) {
02105
                        arithmeticType pi[P::degree], sigma[P::degree];
arithmeticType r = P::template coeff_at_t<P::degree>::template get<arithmeticType>();
02106
02108
                        EFTHorner<P::degree - 1, 0>::func(x, pi, sigma, &r);
02109
                        arithmeticType c = internal::horner<arithmeticType, P::degree - 1>(pi, sigma, x);
02110
                        return r + c;
02111
                   }
02112
               }:
02113
02114
               template<typename coeff, typename... coeffs>
02115
               struct string_helper {
02116
                   static std::string func() {
                        std::string tail = string_helper<coeffs...>::func();
std::string result = "";
02117
02118
02119
                        if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
02120
                             return tail;
02121
                        } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
02122
                            if (sizeof...(coeffs) == 1) {
02123
                                 result += "x";
02124
                             } else {
                                result += "x^" + std::to_string(sizeof...(coeffs));
02125
02126
02127
                        } else {
02128
                            if (sizeof...(coeffs) == 1) {
                                 result += coeff::to_string() + " x";
02129
02130
                            } else {
02131
                                 result += coeff::to_string()
                                          + " x^" + std::to_string(sizeof...(coeffs));
02132
02133
                             }
02134
                        }
02135
02136
                        if (!tail.empty()) {
                            if (tail.at(0) != '-') {
    result += " + " + tail;
02137
02138
02139
                             } else {
02140
                                 result += " - " + tail.substr(1);
02141
02142
                        }
02143
02144
                        return result;
02145
02146
02147
02148
               template<typename coeff>
02149
               struct string helper<coeff> {
02150
                   static std::string func() {
                        if (!std::is_same<coeff, typename Ring::zero>::value) {
02151
02152
                            return coeff::to_string();
02153
                        } else {
                            return "";
02154
02155
02156
                    }
```

```
02157
              };
02158
02159
           public:
              template<typename P>
02162
              using simplify_t = typename simplify<P>::type;
02163
02164
              template<typename v1, typename v2>
02168
02169
              using add_t = typename add<v1, v2>::type;
02170
02174
              template<typename v1, typename v2>
02175
              using sub_t = typename sub<v1, v2>::type;
02176
              template<typename v1, typename v2>
02180
02181
              using mul_t = typename mul<v1, v2>::type;
02182
02186
              template<typename v1, typename v2>
02187
              using eq_t = typename eq_helper<v1, v2>::type;
02188
02192
              template<typename v1, typename v2>
02193
              using lt_t = typename lt_helper<v1, v2>::type;
02194
02198
               template<typename v1, typename v2>
02199
              using gt_t = typename gt_helper<v1, v2>::type;
02200
02204
               template<typename v1, typename v2>
              using div_t = typename div<v1, v2>::q_type;
02205
02206
02210
               template<typename v1, typename v2>
02211
              using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
02212
02216
              template<typename coeff, size_t deg>
02217
              using monomial_t = typename monomial<coeff, deg>::type;
02218
02221
               template<typename v>
02222
              using derive_t = typename derive_helper<v>::type;
02223
02226
              template<typename v>
              using pos_t = typename Ring::template pos_t<typename v::aN>;
02228
02231
              template<typename v>
02232
              static constexpr bool pos_v = pos_t<v>::value;
02233
              template<typename v1, typename v2>
02237
              using gcd_t = std::conditional_t<
02238
02239
                   Ring::is_euclidean_domain,
02240
                   typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
02241
                  void>;
02242
02245
              template<auto x>
02246
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
02247
02250
               template<typename v>
02251
              using inject_ring_t = val<v>;
02252      };
02253 } // namespace aerobus
02254
02255 // fraction field
02256 namespace aerobus {
02257
          namespace internal {
02258
              template<typename Ring, typename E = void>
              requires IsEuclideanDomain<Ring>
02259
              struct _FractionField {};
02260
02261
02262
              template<typename Ring>
02263
              requires IsEuclideanDomain<Ring>
02264
              struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
02266
                  static constexpr bool is_field = true;
static constexpr bool is_euclidean_domain = true;
02267
02268
02269
               private:
02270
                  template<typename val1, typename val2, typename E = void>
02271
                   struct to_string_helper {};
02272
                  template<typename val1, typename val2>
struct to_string_helper <val1, val2,</pre>
02273
02274
02275
                       std::enable_if_t<
02276
                       Ring::template eq_t<
02277
                       val2, typename Ring::one
02278
                       >::value
02279
02280
02281
                       static std::string func()
02282
                           return vall::to_string();
02283
02284
                   } ;
02285
02286
                   template<tvpename val1, tvpename val2>
```

```
struct to_string_helper<val1, val2,
                      std::enable_if_t<
02288
02289
                       !Ring::template eq_t<
02290
                      val2,
02291
                      typename Ring::one
02292
                      >::value
02293
02294
                  > {
02295
                      static std::string func() {
                          return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
02296
02297
02298
                  };
02299
02300
               public:
02304
                  template<typename val1, typename val2>
02305
                  struct val {
                      using x = val1;
02307
                      using y = val2;
02309
02311
                      using is_zero_t = typename val1::is_zero_t;
02313
                      static constexpr bool is_zero_v = val1::is_zero_t::value;
02314
02316
                      using ring_type = Ring;
                      using enclosing_type = _FractionField<Ring>;
02317
02318
02321
                       static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
02322
02326
                      template<typename valueType>
                       static constexpr INLINED DEVICE valueType get() {
02327
02328
                          return internal::staticcast<valueType, typename ring_type::inner_type>::template
      func<x::v>() /
02329
                               internal::staticcast<valueType, typename ring_type::inner_type>::template
      func<v::v>();
02330
02331
02334
                      static std::string to_string() {
02335
                          return to_string_helper<val1, val2>::func();
02336
02337
02342
                      template<typename arithmeticType>
02343
                      static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& v) {
02344
                           return x::eval(v) / y::eval(v);
02345
02346
                  }:
02347
02349
                  using zero = val<typename Ring::zero, typename Ring::one>;
02351
                  using one = val<typename Ring::one, typename Ring::one>;
02352
02355
                  template<typename v>
02356
                  using inject_t = val<v, typename Ring::one>;
02357
02360
                  template<auto x>
                  using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
02361
      Ring::one>;
02362
02365
                  template<typename v>
                  using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
02366
02367
02369
                  using ring_type = Ring;
02370
               private:
02371
02372
                  template<typename v, typename E = void>
02373
                  struct simplify {};
02374
02375
02376
                  template<typename v>
02377
                  struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {</pre>
02378
                      using type = typename _FractionField<Ring>::zero;
02379
                  };
02380
02381
                  // x != 0
02382
                  template<typename v>
02383
                  struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value» {
                   private:
02384
02385
                             _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
                      using .
                      using newx = typename Ring::template div_t<typename v::x, _gcd>;
02386
02387
                      using newy = typename Ring::template div_t<typename v::y, _gcd>;
02388
02389
                      using posx = std::conditional_t<
02390
                                           !Ring::template pos_v<newy>,
                                           typename Ring::template sub_t<typename Ring::zero, newx>,
02391
02392
                                           newx>;
02393
                      using posy = std::conditional_t<
02394
                                           !Ring::template pos_v<newy>,
02395
                                           typename Ring::template sub_t<typename Ring::zero, newy>,
02396
                                           newy>;
02397
                   public:
02398
                      using type = typename FractionField<Ring>::template val<posx, posy>;
```

```
02399
                               };
02400
02401
                          public:
02404
                               template<typename v>
02405
                               using simplify_t = typename simplify<v>::type;
02406
02407
02408
                               template<typename v1, typename v2>
02409
                                struct add {
02410
                                 private:
02411
                                      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>;
02412
02413
                                       using dividend = typename Ring::template add_t<a, b>;
02414
                                      using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02415
                                      using g = typename Ring::template gcd_t<dividend, diviser>;
02416
                                 public:
02417
                                       using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
02418
         diviser»;
02419
                                };
02420
02421
                               template<typename v>
02422
                               struct pos {
02423
                                       using type = std::conditional t<
02424
                                              (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
                                              (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
02425
02426
02427
                                             std::false_type>;
02428
                               };
02429
02430
                               template<typename v1, typename v2>
02431
                               struct sub {
02432
                                private:
                                      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>;
02433
02434
02435
                                      using dividend = typename Ring::template sub_t<a, b>;
02436
                                      using diviser = typename Ring::template mul t<typename v1::v, typename v2::v>;
                                      using g = typename Ring::template gcd_t<dividend, diviser>;
02437
02438
02439
                                 public:
02440
                                       using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
          diviser»:
02441
                                }:
02442
02443
                               template<typename v1, typename v2>
02444
                                struct mul
                                 private:
02445
                                      using a = typename Ring::template mul_t<typename v1::x, typename v2::x>; using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02446
02447
02448
02449
                                 public:
02450
                                      using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
02451
02452
02453
                               template<typename v1, typename v2, typename E = void>
02454
                               struct div {};
02455
02456
                               template<typename v1, typename v2>
_FractionField<Ring>::zero>::value>
02458
                                struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
02459
                                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02460
                                      using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02461
                                 public:
02462
02463
                                      using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
02464
                               };
02465
02466
                               template<tvpename v1, tvpename v2>
                               struct div<v1, v2, std::enable_if_t<
02467
02468
                                     std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
02469
                                       using type = one;
02470
                               };
02471
02472
                               template<typename v1, typename v2>
02473
                               struct eq {
02474
                                      using type = std::conditional_t<
02475
                                                    std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
02476
                                                     \verb|std::is_same<| typename | simplify_t < v1>::y, | typename | simplify_t < v2>::y>::value, | typename | type
02477
                                             std::true_type,
02478
                                             std::false_type>;
02479
                               };
02480
02481
                               template<typename v1, typename v2, typename E = void>
02482
                               struct gt;
02483
02484
                               template<tvpename v1, tvpename v2>
```

```
struct gt<v1, v2, std::enable_if_t<
02486
                       (eq<v1, v2>::type::value)
02487
02488
                       using type = std::false_type;
02489
02490
02491
                  template<typename v1, typename v2>
02492
                  struct gt<v1, v2, std::enable_if_t<
02493
                      (!eq<v1, v2>::type::value) &&
02494
                       (!pos<v1>::type::value) && (!pos<v2>::type::value)
02495
02496
                      using type = typename gt<
02497
                           typename sub<zero, v1>::type, typename sub<zero, v2>::type
02498
                       >::type;
02499
                  };
02500
02501
                  template<typename v1, typename v2>
                  struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02502
02504
                       (pos<v1>::type::value) && (!pos<v2>::type::value)
02505
02506
                       using type = std::true_type;
02507
                  };
02508
02509
                  template<typename v1, typename v2>
                  struct gt<v1, v2, std::enable_if_t<
02510
02511
                       (!eq<v1, v2>::type::value) &&
02512
                       (!pos<v1>::type::value) && (pos<v2>::type::value)
02513
02514
                       using type = std::false_type;
02515
                  };
02516
02517
                  template<typename v1, typename v2>
02518
                  struct gt<v1, v2, std::enable_if_t<
02519
                       (!eq<v1, v2>::type::value) &&
02520
                       (pos<v1>::type::value) && (pos<v2>::type::value)
02521
                       using type = typename Ring::template gt_t<
02523
                           typename Ring::template mul_t<v1::x, v2::y>,
02524
                           typename Ring::template mul_t<v2::y, v2::x>
02525
02526
                  };
02527
02528
               public:
                  template<typename v1, typename v2>
02532
02533
                  using add_t = typename add<v1, v2>::type;
02534
02539
                  template<typename v1, typename v2>
02540
                  using mod_t = zero;
02541
                  template<typename v1, typename v2>
02547
                  using gcd_t = v1;
02548
02552
                  template<typename v1, typename v2>
02553
                  using sub_t = typename sub<v1, v2>::type;
02554
02558
                  template<typename v1, typename v2>
02559
                  using mul_t = typename mul<v1, v2>::type;
02560
02564
                  template<typename v1, typename v2>
02565
                  using div_t = typename div<v1, v2>::type;
02566
                  template<typename v1, typename v2>
02571
                  using eq_t = typename eq<v1, v2>::type;
02572
02576
                  template<typename v1, typename v2>
                  static constexpr bool eq_v = eq<v1, v2>::type::value;
02577
02578
                  template<typename v1, typename v2>
02582
                  using gt_t = typename gt<v1, v2>::type;
02584
02588
                  template<typename v1, typename v2>
02589
                  static constexpr bool gt_v = gt<v1, v2>::type::value;
02590
02593
                  template<typename v1>
02594
                  using pos_t = typename pos<v1>::type;
02595
02598
                  template<typename v>
02599
                  static constexpr bool pos_v = pos_t<v>::value;
02600
              }:
02601
02602
              template<typename Ring, typename E = void>
02603
              requires IsEuclideanDomain<Ring>
02604
              struct FractionFieldImpl {};
02605
              // fraction field of a field is the field itself
02606
02607
              template<typename Field>
```

```
requires IsEuclideanDomain<Field>
                      struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {</pre>
02609
02610
                             using type = Field;
02611
                            template<typename v>
02612
                             using inject_t = v;
02613
                      };
02614
02615
                      \ensuremath{//} fraction field of a ring is the actual fraction field
02616
                       template<typename Ring>
02617
                      requires IsEuclideanDomain<Ring>
                      struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field> {
02618
02619
                            using type = _FractionField<Ring>;
02620
02621
               } // namespace internal
02622
02625
               template<typename Ring>
                requires IsEuclideanDomain<Ring>
02626
                using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02627
02628
02631
                template<typename Ring>
02632
                struct Embed<Ring, FractionField<Ring» {</pre>
02635
                      template<typename v>
                      using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02636
02637
02638 } // namespace aerobus
02639
02640
02641 // short names for common types
02642 namespace aerobus {
02646
                template<typename X, typename Y>
02647
                requires IsRing<typename X::enclosing_type> &&
02648
                      (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02649
                using add_t = typename X::enclosing_type::template add_t<X, Y>;
02650
02654
                template<typename X, typename Y>
                requires IsRing<typename X::enclosing_type> &&
02655
                      (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02656
02657
                using sub_t = typename X::enclosing_type::template sub_t<X, Y>;
02658
02662
                template<typename X, typename Y>
02663
                requires IsRing<typename X::enclosing_type> &&
02664
                      (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02665
                using mul_t = typename X::enclosing_type::template mul_t<X, Y>;
02666
02670
                template<typename X, typename Y>
02671
                requires IsEuclideanDomain<typename X::enclosing_type> &&
02672
                       (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02673
                using div_t = typename X::enclosing_type::template div_t<X, Y>;
02674
02677
                using g32 = FractionField<i32>;
02678
02681
                using fpq32 = FractionField<polynomial<q32>>;
02682
02685
                using g64 = FractionField<i64>;
02686
02688
                using pi64 = polynomial<i64>;
02689
02691
                using pq64 = polynomial<q64>;
02692
02694
                using fpq64 = FractionField<polynomial<q64>>;
02695
02700
                template<typename Ring, typename v1, typename v2>
02701
                using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02702
02709
                template<typename v>
02710
                using embed_int_poly_in_fractions_t =
                             typename Embed<
02711
02712
                                   polynomial<typename v::ring_type>,
02713
                                   polynomial<FractionField<typename v::ring_type>»::template type<v>;
02714
02718
                template<int64_t p, int64_t q>
02719
                using make_q64_t = typename q64::template simplify_t<
02720
                                    typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02721
               template<int32_t p, int32_t q>
using make_q32_t = typename q32::template simplify_t
02725
02726
02727
                                    typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>»;
02728
                \label{template} $$ \ensuremath{\mbox{typename Ring, typename v1, typename v2>} $$ using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>; $$ \ensuremath{\mbox{v2}}$ is $$ \ensuremath{\mbox{ring}}$ is $$ \ensuremath{\mbox{v2}}$ is $$ \ensuremath{\mbox{v
02733
02734
02739
                template<typename Ring, typename v1, typename v2>
02740
                using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02741
02743
                template<>
02744
                struct Embed<q32, q64> {
02747
                      template<typename v>
02748
                      using type = make_g64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
```

```
02749
          };
02750
02754
          template<typename Small, typename Large>
02755
          struct Embed<polynomial<Small>, polynomial<Large» {</pre>
          private:
02756
02757
              template<typename v, typename i>
02758
              struct at_low;
02759
02760
              template<typename v, size_t i>
02761
              struct at_index {
                 using type = typename Embed<Small, Large>::template
02762
     type<typename v::template coeff_at_t<i>>;
02763
              };
02764
02765
              template<typename v, size_t... Is>
02766
              struct at_low<v, std::index_sequence<Is...» {</pre>
02767
                  using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
02768
              };
02769
02770
          public:
02773
              template<typename v>
02774
              using type = typename at_low<v, typename internal::make_index_sequence_reverse<v::degree +</pre>
     1»::type;
02775
         };
02776
02780
          template<typename Ring, auto... xs>
02781
          using make_int_polynomial_t = typename polynomial<Ring>::template val<</pre>
02782
                  typename Ring::template inject_constant_t<xs>...>;
02783
02787
          template<typename Ring, auto... xs>
         using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<</pre>
02788
02789
                  typename FractionField<Ring>::template inject_constant_t<xs>...>;
02790 } // namespace aerobus
02791
02792 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02793 namespace aerobus {
02794
         namespace internal {
             template<typename T, size_t x, typename E = void>
02796
              struct factorial {};
02797
02798
             template<typename T, size_t x>
02799
              struct factorial<T, x, std::enable_if_t<(x > 0)  {
02800
              private:
02801
                  template<typename, size_t, typename>
02802
                  friend struct factorial;
02803
              public:
02804
                 using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
     x - 1>::type>;
02805
                  static constexpr typename T::inner type value = type::template get<typename
     T::inner_type>();
02806
             };
02807
02808
             template<typename T>
02809
              struct factorial<T, 0> {
02810
              public:
                 using type = typename T::one;
02811
                  static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
02813
02814
          } // namespace internal
02815
02819
          template<typename T, size_t i>
02820
          using factorial_t = typename internal::factorial<T, i>::type;
02821
02825
          template<typename T, size_t i>
02826
          inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02827
02828
          namespace internal {
02829
              template<typename T, size_t k, size_t n, typename E = void>
              struct combination_helper {};
02831
02832
              template<typename T, size_t k, size_t n>
              struct combination_helperTT, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
    using type = typename FractionField<T>::template mul_t<
        typename combination_helper<T, k - 1, n - 1>::type,
02833
02834
02835
02836
                      makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02837
              };
02838
02839
              template<typename T, size_t k, size_t n>
              02840
02841
                  using type = typename combination_helper<T, n - k, n>::type;
02842
              };
02843
02844
              template<typename T, size_t n>
02845
              struct combination_helper<T, 0, n> {
02846
                  using type = typename FractionField<T>::one;
02847
              };
```

```
02849
              template<typename T, size_t k, size_t n>
02850
              struct combination {
02851
                  using type = typename internal::combination_helper<T, k, n>::type::x;
02852
                  02853
      T::inner_type>();
02854
02855
          } // namespace internal
02856
          template<typename T, size_t k, size_t n>
using combination_t = typename internal::combination<T, k, n>::type;
02859
02860
02861
02866
          template<typename T, size_t k, size_t n>
02867
          inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02868
02869
          namespace internal {
              template<typename T, size_t m>
struct bernoulli;
02870
02871
02872
02873
              template<typename T, typename accum, size_t k, size_t m>
02874
              struct bernoulli_helper {
                  using type = typename bernoulli_helper<
02875
02876
02877
                      addfractions_t<T,
02878
                          accum,
02879
                           mulfractions_t<T,</pre>
02880
                              makefraction_t<T,
02881
                                  combination_t<T, k, m + 1>,
02882
                                   typename T::one>,
02883
                               typename bernoulli<T, k>::type
02884
02885
                      k + 1,
02886
                      m>::type;
02887
02888
              };
02889
              template<typename T, typename accum, size_t m>
02891
              struct bernoulli_helper<T, accum, m, m> {
02892
                  using type = accum;
02893
02894
02895
02896
02897
              template<typename T, size_t m>
02898
              struct bernoulli {
02899
                  using type = typename FractionField<T>::template mul_t<</pre>
02900
                      typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02901
                      makefraction t<T.
02902
                      typename T::template val<static_cast<typename T::inner_type>(-1)>,
02903
                      typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02904
02905
02906
02907
                  template<typename floatType>
02908
                  static constexpr floatType value = type::template get<floatType>();
02909
02910
02911
              template<typename T>
02912
              struct bernoulli<T, 0> {
                  using type = typename FractionField<T>::one;
02913
02914
02915
                  template<typename floatType>
02916
                  static constexpr floatType value = type::template get<floatType>();
02917
              };
02918
          } // namespace internal
02919
02923
          template<typename T, size_t n>
02924
          using bernoulli_t = typename internal::bernoulli<T, n>::type;
02925
          template<typename FloatType, typename T, size_t n >
inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02930
02931
02932
          // bell numbers
02933
02934
          namespace internal {
02935
              template<typename T, size_t n, typename E = void>
02936
              struct bell_helper;
02937
02938
              template <typename T, size_t n>
              struct bell_helper<T, n, std::enable_if_t<(n > 1)» {
02939
02940
                  template<typename accum, size_t i, size_t stop>
02941
                  struct sum_helper {
02942
                   private:
02943
                      using left = typename T::template mul_t<
02944
                                   combination\_t < T, i, n-1 >,
                                   typename bell_helper<T, i>::type>;
02945
02946
                      using new_accum = typename T::template add_t<accum, left>;
```

```
public:
02948
                      using type = typename sum_helper<new_accum, i+1, stop>::type;
02949
                   };
02950
02951
                  template<typename accum, size_t stop>
                  struct sum_helper<accum, stop, stop> {
02952
02953
                      using type = accum;
02954
02955
02956
                  using type = typename sum_helper<typename T::zero, 0, n>::type;
              };
02957
02958
              template<typename T>
02959
02960
              struct bell_helper<T, 0> {
02961
                  using type = typename T::one;
02962
02963
02964
              template<typename T>
              struct bell_helper<T, 1> {
02965
02966
                  using type = typename T::one;
02967
02968
          } // namespace internal
02969
          template<typename T, size_t n>
02973
02974
          using bell_t = typename internal::bell_helper<T, n>::type;
02975
02979
          template<typename T, size_t n>
02980
          static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
02981
02982
          namespace internal {
02983
              template<typename T, int k, typename E = void>
02984
              struct alternate { };
02985
02986
              template<typename T, int k>
02987
              struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
                 using type = typename T::one;
02988
02989
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02990
              };
02991
              template<typename T, int k>
struct alternate<T, k, std::enable_if_t<k % 2 != 0» {</pre>
02992
02993
                  using type = typename T::template sub_t<typename T::zero, typename T::one>;
02994
                   static constexpr typename T::inner_type value = type::template get<typename
02995
     T::inner_type>();
02996
02997
          } // namespace internal
02998
03001
          template<typename T, int k>
03002
          using alternate t = typename internal::alternate<T, k>::type;
03003
03006
          template<typename T, size_t k>
03007
          inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
03008
03009
          namespace internal {
              template<typename T, int n, int k, typename E = void>
03010
              struct stirling_1_helper {};
03011
03012
03013
              template<typename T>
03014
              struct stirling_1_helper<T, 0, 0> {
03015
                  using type = typename T::one;
03016
              };
03017
03018
              template<typename T, int n>
03019
              struct stirling_1_helper<T, n, 0, std::enable_if_t<(n > 0)» {
03020
                 using type = typename T::zero;
03021
03022
03023
              template<typename T, int n>
              struct stirling_1_helper<T, 0, n, std::enable_if_t<(n > 0)» {
03024
03025
                 using type = typename T::zero;
03026
03027
              template<typename T, int n, int k>
03028
              struct stirling_1_helperTT, n, k, std::enable_if_t<(k > 0) && (n > 0)» {
    using type = typename T::template sub_t<
03029
03030
03031
                                    typename stirling_1_helper<T, n-1, k-1>::type,
03032
                                    typename T::template mul_t<
03033
                                        typename T::template inject_constant_t<n-1>,
03034
                                        typename stirling_1_helper<T, n-1, k>::type
03035
03036
              };
03037
          } // namespace internal
03038
          template<typename T, int n, int k>
using stirling_1_signed_t = typename internal::stirling_1_helper<T, n, k>::type;
03043
03044
03045
```

```
template<typename T, int n, int k>
03051
           using stirling_1_unsigned_t = abs_t<typename internal::stirling_1_helper<T, n, k>::type>;
03052
03057
           template<typename T, int n, int k>
03058
           static constexpr typename T::inner_type stirling_1_unsigned_v = stirling_1_unsigned_t<T, n, k>::v;
03059
03064
           template<typename T, int n, int k>
03065
           static constexpr typename T::inner_type stirling_1_signed_v = stirling_1_signed_t<T, n, k>::v;
03066
03067
           namespace internal {
               template<typename T, int n, int k, typename E = void>
03068
03069
               struct stirling_2_helper {};
03070
03071
               template<typename T, int n>
03072
               struct stirling_2_helper<T, n, n, std::enable_if_t<(n >= 0)» {
03073
                   using type = typename T::one;
03074
               };
03075
03076
               template<typename T, int n>
03077
               struct stirling_2_helper<T, n, 0, std::enable_if_t<(n > 0)» {
03078
                   using type = typename T::zero;
03079
03080
03081
               template<typename T, int n>
03082
               struct stirling_2_helper<T, 0, n, std::enable_if_t<(n > 0)» {
                 using type = typename T::zero;
03083
03084
03085
               03086
03087
03088
03089
                                     typename stirling_2_helper<T, n-1, k-1>::type,
03090
                                     typename T::template mul_t<
03091
                                         typename T::template inject_constant_t<k>,
03092
                                         typename stirling_2_helper<T, n-1, k>::type
03093
03094
           };
} // namespace internal
03095
03096
03101
           template<typename T, int n, int k>
03102
           using stirling_2_t = typename internal::stirling_2_helper<T, n, k>::type;
03103
          template<typename T, int n, int k>
static constexpr typename T::inner_type stirling_2_v = stirling_2_t<T, n, k>::v;
03108
03109
03110
03111
           namespace internal {
03112
               template<typename T>
03113
               struct pow_scalar {
03114
                   template<size_t p>
                   static constexpr DEVICE INLINED T func(const T& x) { return p == 0 ? static_cast<T>(1) :
03115
                       p % 2 == 0 ? func<p/2>(x) * func<p/2>(x) :
03116
03117
                        x * func<p/2>(x) * func<p/2>(x);
03118
03119
               };
03120
               template<typename T, typename p, size_t n, typename E = void>
03121
               requires IsEuclideanDomain<T>
03122
03123
               struct pow;
03124
03125
               template<typename T, typename p, size_t n>
               struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)» {
    using type = typename T::template mul_t<
03126
03127
03128
                       typename pow<T, p, n/2>::type,
03129
                        typename pow<T, p, n/2>::type
03130
03131
              };
03132
               template<typename T, typename p, size_t n>
struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)» {
    using type = typename T::template mul_t</pre>
03133
03134
03135
03136
                       p,
03137
                        typename T::template mul_t<
                            typename pow<T, p, n/2>::type, typename pow<T, p, n/2>::type
03138
03139
03140
03141
                   >;
03142
03143
03144
               template<typename T, typename p, size_t n>
               struct pow<T, p, n, std::enable_if_t<n == 0» { using type = typename T::one; };</pre>
03145
          } // namespace internal
03146
03147
          template<typename T, typename p, size_t n>
using pow_t = typename internal::pow<T, p, n>::type;
03152
03153
03154
          template<typename T, typename p, size_t n>
static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
03159
03160
```

```
03161
03162
          template<typename T, size_t p>
03163
          static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
      internal::pow_scalar<T>::template func(x); }
03164
03165
          namespace internal {
03166
              template<typename, template<typename, size_t> typename, class>
03167
              struct make_taylor_impl;
03168
03169
              template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
              struct make_taylor_implcT, coeff_at, std::integer_sequence<size_t, Is...» {
    using type = typename polynomial<FractionField<T>::template val<typename coeff_at<T,
03170
03171
      Is>::type...>;
03172
              } ;
03173
0.3174
03179
          template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
03180
          using taylor = typename internal::make_taylor_impl<</pre>
03181
03182
               coeff_at,
03183
               internal::make_index_sequence_reverse<deg + 1>>::type;
03184
03185
          namespace internal {
               template<typename T, size_t i>
03186
03187
               struct exp_coeff {
03188
                 using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03189
03190
0.3191
               template<typename T, size_t i, typename E = void>
03192
               struct sin_coeff_helper {};
03193
03194
               template<typename T, size t i>
03195
               struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03196
                   using type = typename FractionField<T>::zero;
03197
03198
03199
               template<typename T, size_t i>
              struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03200
                  using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03201
03202
03203
03204
               template<typename T, size_t i>
03205
               struct sin_coeff {
03206
                  using type = typename sin_coeff_helper<T, i>::type;
03207
03208
03209
               template<typename T, size_t i, typename E = void>
03210
               struct sh_coeff_helper {};
03211
03212
               template<typename T, size_t i>
03213
               struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03214
                   using type = typename FractionField<T>::zero;
03215
03216
               template<typename T, size_t i>
03217
              struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
    using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03218
03219
03220
03221
03222
               template<typename T, size_t i>
03223
               struct sh_coeff {
03224
                   using type = typename sh_coeff_helper<T, i>::type;
03225
03226
03227
               template<typename T, size_t i, typename E = void>
03228
               struct cos_coeff_helper {};
03229
03230
               template<typename T, size_t i>
03231
              struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
                  using type = typename FractionField<T>::zero;
03232
03233
03234
03235
               template<typename T, size_t i>
               struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03236
03237
                  using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03238
03239
03240
               template<typename T, size_t i>
03241
               struct cos_coeff {
03242
                  using type = typename cos_coeff_helper<T, i>::type;
03243
03244
03245
               template<typename T, size_t i, typename E = void>
03246
               struct cosh_coeff_helper {};
03247
              template<typename T, size_t i>
struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03248
03249
```

```
using type = typename FractionField<T>::zero;
03251
03252
03253
              template<typename T, size_t i>
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03254
03255
03256
03257
03258
              template<typename T, size_t i>
03259
              struct cosh coeff {
03260
                  using type = typename cosh_coeff_helper<T, i>::type;
03261
03262
03263
              template<typename T, size_t i>
03264
              struct geom_coeff { using type = typename FractionField<T>::one; };
03265
03266
03267
              template<typename T, size_t i, typename E = void>
03268
              struct atan_coeff_helper;
03269
03270
              template<typename T, size_t i>
03271
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03272
                 using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;;
03273
03274
03275
              template<typename T, size_t i>
03276
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03277
                  using type = typename FractionField<T>::zero;
03278
03279
03280
              template<tvpename T, size t i>
03281
              struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
03282
03283
              template<typename T, size_t i, typename E = void>
03284
              struct asin_coeff_helper;
03285
03286
              template<typename T, size t i>
              struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03287
03288
                  using type = makefraction_t<T,
03289
                       factorial_t<T, i - 1>,
03290
                       typename T::template mul_t<
03291
                           typename T::template val<i>,
03292
                           T::template mul t<
03293
                               pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
03294
                               pow<T, factorial_t<T, i / 2>, 2
03295
03296
                       >
03297
                       »;
03298
              };
03299
03300
              template<typename T, size_t i>
03301
              struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03302
                  using type = typename FractionField<T>::zero;
03303
03304
03305
              template<typename T, size_t i>
03306
              struct asin_coeff {
                  using type = typename asin_coeff_helper<T, i>::type;
03307
03308
03309
              template<typename T, size_t i>
03310
03311
              struct lnp1_coeff {
03312
                  using type = makefraction_t<T,
03313
                      alternate_t<T, i + 1>,
03314
                       typename T::template val<i>;;
03315
              } ;
03316
03317
              template<tvpename T>
03318
              struct lnpl_coeff<T, 0> { using type = typename FractionField<T>::zero; };
03319
03320
              template<typename T, size_t i, typename E = void>
03321
              struct asinh_coeff_helper;
03322
03323
              template<tvpename T, size t i>
03324
              struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03325
                  using type = makefraction_t<T,
03326
                       typename T::template mul_t<
03327
                           alternate_t<T, i / 2>,
03328
                           factorial_t<T, i - 1>
03329
03330
                       typename T::template mul_t<</pre>
03331
                           typename T::template mul_t<
03332
                               typename T::template val<i>,
                               pow_t<T, factorial_t<T, i / 2>, 2>
03333
03334
                           pow_t<T, typename T::template inject_constant_t<4>, i / 2>
03335
03336
```

```
>;
03338
03339
03340
               template<typename T, size_t i>
               struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
03341
03342
03343
03344
03345
               template<typename T, size_t i>
03346
               struct asinh coeff {
                    using type = typename asinh_coeff_helper<T, i>::type;
03347
03348
03349
03350
               template<typename T, size_t i, typename E = void>
03351
               struct atanh_coeff_helper;
03352
03353
               template<typename T, size_t i>
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
    // 1/i</pre>
03354
03355
03356
                    using type = typename FractionField<T>:: template val<
03357
                        typename T::one,
03358
                        typename T::template inject_constant_t<i>;;
03359
               };
03360
03361
               template<typename T, size_t i>
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03362
                    using type = typename FractionField<T>::zero;
03363
03364
03365
               template<typename T, size_t i>
03366
03367
               struct atanh_coeff {
03368
                   using type = typename atanh_coeff_helper<T, i>::type;
03369
03370
03371
               template<typename T, size_t i, typename E = void>
03372
               struct tan_coeff_helper;
03373
03374
                template<typename T, size_t i>
               struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
    using type = typename FractionField<T>::zero;
03375
03376
03377
03378
               template<typename T, size_t i>
struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
03379
03380
03381
               private:
03382
                    // 4^((i+1)/2)
                    using _4p = typename FractionField<T>::template inject_t<
    pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
// 4^((i+1)/2) - 1
03383
03384
03385
                    using _4pm1 = typename FractionField<T>::template
03386
      sub_t<_4p, typename FractionField<T>::one>;
03387
                    // (-1)^((i-1)/2)
03388
                    using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
03389
                    using dividend = typename FractionField<T>::template mul_t<</pre>
03390
                         altp,
03391
                        FractionField<T>::template mul t<
03392
                        _4p,
03393
                         FractionField<T>::template mul_t<
03394
                        _4pm1,
03395
                         bernoulli_t < T, (i + 1) >
03396
03397
03398
03399
                public:
03400
                    using type = typename FractionField<T>::template div_t<dividend,</pre>
03401
                        typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03402
               };
03403
03404
               template<typename T, size t i>
03405
               struct tan_coeff {
03406
                   using type = typename tan_coeff_helper<T, i>::type;
03407
03408
               template<typename T, size_t i, typename E = void>
03409
03410
               struct tanh coeff helper;
03411
03412
                template<typename T, size_t i>
03413
                struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {</pre>
03414
                    using type = typename FractionField<T>::zero;
03415
03416
               template<typename T, size_t i>
03418
                struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
               private:
03419
03420
                    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
03421
03422
```

```
sub_t<_4p, typename FractionField<T>::one>;
                  using dividend =
03423
                      typename FractionField<T>::template mul_t<</pre>
03424
                          _4p,
03425
                          typename FractionField<T>::template mul_t<</pre>
03426
03427
                               4pm1.
03428
                              bernoulli_t<T, (i + 1) >>::type;
03429
              public:
03430
              using type = typename FractionField<T>::template div_t<dividend,
03431
                      FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03432
             };
03433
03434
              template<typename T, size_t i>
03435
              struct tanh_coeff {
03436
                 using type = typename tanh_coeff_helper<T, i>::type;
03437
         } // namespace internal
03438
03439
03443
          template<typename Integers, size_t deg>
03444
          using exp = taylor<Integers, internal::exp_coeff, deg>;
03445
03449
          template<typename Integers, size_t deg>
          using expm1 = typename polynomial<FractionField<Integers>>::template sub_t<</pre>
03450
              exp<Integers, deg>
03451
03452
              typename polynomial<FractionField<Integers>>::one>;
03453
03457
          template<typename Integers, size_t deg>
03458
          using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
03459
03463
          template<typename Integers, size_t deg>
03464
          using atan = taylor<Integers, internal::atan coeff, deg>;
03465
03469
          template<typename Integers, size_t deg>
03470
          using sin = taylor<Integers, internal::sin_coeff, deg>;
03471
03475
          template<typename Integers, size_t deg>
03476
          using sinh = taylor<Integers, internal::sh_coeff, deg>;
03477
03482
          template<typename Integers, size_t deg>
03483
          using cosh = taylor<Integers, internal::cosh_coeff, deg>;
03484
03489
          template<typename Integers, size_t deg>
03490
          using cos = taylor<Integers, internal::cos_coeff, deg>;
03491
03496
          template<typename Integers, size_t deg>
03497
          using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
03498
03503
          template<typename Integers, size_t deg>
03504
          using asin = taylor<Integers, internal::asin_coeff, deg>;
03505
03510
          template<typename Integers, size_t deg>
03511
          using asinh = taylor<Integers, internal::asinh_coeff, deg>;
03512
03517
          template<typename Integers, size_t deg>
03518
          using atanh = taylor<Integers, internal::atanh_coeff, deg>;
03519
03524
          template<typename Integers, size_t deg>
03525
          using tan = taylor<Integers, internal::tan_coeff, deg>;
03526
03531
          template<typename Integers, size_t deg>
         using tanh = taylor<Integers, internal::tanh_coeff, deg>;
03532
03533 } // namespace aerobus
03534
03535 // continued fractions
03536 namespace aerobus {
03539
         template<int64_t... values>
03540
         struct ContinuedFraction {};
03541
03544
          template<int64 t a0>
03545
          struct ContinuedFraction<a0> {
03547
             using type = typename q64::template inject_constant_t<a0>;
03549
              static constexpr double val = static_cast<double>(a0);
03550
         };
03551
03555
          template<int64_t a0, int64_t... rest>
03556
          struct ContinuedFraction<a0, rest...> {
03558
             using type = q64::template add_t<
03559
                      typename q64::template inject_constant_t<a0>,
03560
                      typename q64::template div_t<
03561
                          typename q64::one,
                          typename ContinuedFraction<rest...>::type
03562
03563
03564
03566
              static constexpr double val = type::template get<double>();
03567
          };
03568
03572
          using PI_fraction =
```

```
ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1>;
03574
               using E_fraction =
         ContinuedFraction<2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1>;
               using SQRT2_fraction =
03576
         using SQRT3_fraction =
03578
          ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 
           // NOLINT
03579 } // namespace aerobus
03580
03581 // known polynomials
03582 namespace aerobus {
03583
                 // CChebyshev
03584
                 namespace internal {
03585
                       template<int kind, size_t deg, typename I>
03586
                        struct chebyshev_helper {
03587
                              using type = typename polynomial<I>::template sub_t<</pre>
                                     typename polynomial<I>::template mul_t<
03588
                                            typename polynomial<I>::template mul_t<
03589
03590
                                                   typename polynomial<I>::template inject_constant_t<2>,
03591
                                                   typename polynomial<I>::X>,
03592
                                            typename chebyshev_helper<kind, deg - 1, I>::type
03593
03594
                                     typename chebyshev_helper<kind, deg - 2, I>::type
03595
                              >;
03596
                       };
03597
03598
                       template<typename I>
03599
                       struct chebyshev_helper<1, 0, I> {
03600
                              using type = typename polynomial<I>::one;
03601
03602
03603
                       template<typename I>
03604
                       struct chebyshev_helper<1, 1, I> {
03605
                              using type = typename polynomial<I>::X;
03606
03607
03608
                        template<typename I>
03609
                       struct chebyshev_helper<2, 0, I> {
03610
                            using type = typename polynomial<I>::one;
03611
03612
03613
                       template<tvpename T>
03614
                       struct chebyshev_helper<2, 1, I> {
                              using type = typename polynomial<I>::template mul_t<</pre>
03615
03616
                                     typename polynomial<I>::template inject_constant_t<2>,
03617
                                     typename polynomial<I>::X>;
03618
                 } // namespace internal
03619
03620
03621
                 // Laguerre
03622
                 namespace internal {
03623
                       template<size_t deg, typename I>
03624
                        struct laguerre_helper {
                              using Q = FractionField<I>;
03625
                              using PQ = polynomial<Q>;
03626
03627
03628
03629
                               // Lk = (1 / k) * ((2 * k - 1 - x) * 1km1 - (k - 2) Lkm2)
                              using lnm2 = typename laguerre_helper<deg - 2, I>::type;
03630
                              using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03631
                               // -x + 2k-1
03632
03633
                              using p = typename PQ::template val<
                                   typename Q::template inject_constant_t<-1>,
03634
03635
                                     typename Q::template inject_constant_t<2 * deg - 1»;
                               // 1/n
03636
03637
                              using factor = typename PQ::template inject_ring_t<</pre>
                                     typename Q::template val<typename I::one, typename I::template
03638
         inject constant t<deg>>:
03639
03640
                         public:
03641
                              using type = typename PQ::template mul_t <</pre>
03642
                                     factor,
                                     typename PQ::template sub_t<
03643
03644
                                            typename PO::template mul t<
03645
                                                   p,
03646
03647
03648
                                            typename PQ::template mul_t<</pre>
03649
                                                   typename PO::template inject constant t<deg-1>,
03650
                                                   lnm2
03651
03652
03653
                              >;
03654
                       };
03655
03656
                       template<tvpename I>
```

```
struct laguerre_helper<0, I> {
03658
                 using type = typename polynomial<FractionField<I»::one;
03659
              };
03660
03661
              template<typename I>
              struct laquerre_helper<1, I> {
03662
03663
              private:
03664
                  using PQ = polynomial<FractionField<I>;
                public:
03665
03666
                  using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
              };
03667
          } // namespace internal
03668
03669
03670
          // Bernstein
03671
          namespace internal {
03672
              template<size_t i, size_t m, typename I, typename E = void>
03673
              struct bernstein_helper {};
03674
03675
              template<typename I>
              struct bernstein_helper<0, 0, I> {
03676
03677
                 using type = typename polynomial<I>::one;
03678
03679
              template<size_t i, size_t m, typename I>
struct bernstein_helper<i, m, I, std::enable_if_t<</pre>
03680
03681
                         (m > 0) && (i == 0)   {
03682
               private:
03683
03684
                  using P = polynomial<I>;
03685
               public:
03686
                  using type = typename P::template mul_t<
                           typename P::template sub_t<typename P::one, typename P::X>,
03687
03688
                           typename bernstein_helper<i, m-1, I>::type>;
03689
03690
03691
              template<size_t i, size_t m, typename I>
              03692
03693
03694
03695
                  using P = polynomial<I>;
03696
               public:
03697
                  using type = typename P::template mul_t<</pre>
                           typename P::X,
03698
03699
                           typename bernstein helper<i-1, m-1, I>::type>;
03700
              };
03701
03702
              template<size_t i, size_t m, typename I>
              struct bernstein_helper<i, m, I, std::enable_if_t<  (m > 0) \&\& (i > 0) \&\& (i < m) * \{
03703
03704
03705
               private:
                  using P = polynomial<I>;
03706
               public:
03707
03708
                  using type = typename P::template add_t<</pre>
03709
                           typename P::template mul_t<</pre>
03710
                               typename P::template sub_t<typename P::one, typename P::X>,
03711
                               typename bernstein_helper<i, m-1, I>::type>,
03712
                           typename P::template mul t<
03713
                               typename P::X,
03714
                               typename bernstein_helper<i-1, m-1, I>::type»;
03715
          } // namespace internal
03716
03717
          // AllOne polynomials
namespace internal {
03718
03719
03720
             template<size_t deg, typename I>
03721
              struct AllOneHelper {
03722
                  using type = aerobus::add_t<
03723
                      typename polynomial<I>::one,
03724
                      typename aerobus::mul t<
03725
                           typename polynomial<I>::X,
                           typename AllOneHelper<deg-1, I>::type
03726
03727
03728
              } ;
03729
              template<typename I>
03730
03731
              struct AllOneHelper<0, I> {
03732
                using type = typename polynomial<I>::one;
03733
03734
          } // namespace internal
03735
03736
          // Bessel polynomials
03737
          namespace internal {
   template<size_t deg, typename I>
03738
03739
              struct BesselHelper {
               private:
03740
03741
                  using P = polynomial<I>;
03742
                  using factor = typename P::template monomial t<
03743
                       typename I::template inject_constant_t<(2*deg - 1)>,
```

```
03744
                      1>;
03745
               public:
03746
                  using type = typename P::template add_t<
03747
                       typename P::template mul_t<</pre>
03748
                          factor,
03749
                          typename BesselHelper<deg-1, I>::type
03750
03751
                       typename BesselHelper<deg-2, I>::type
03752
03753
              } ;
03754
03755
              template<typename I>
03756
              struct BesselHelper<0, I> {
03757
                  using type = typename polynomial<I>::one;
03758
03759
03760
              template<typename I>
03761
              struct BesselHelper<1, I> {
03762
              private:
03763
                  using P = polynomial<I>;
03764
               public:
03765
                  using type = typename P::template add_t<</pre>
03766
                      typename P::one,
03767
                      typename P::X
03768
03769
              };
03770
          } // namespace internal
03771
03772
          namespace known_polynomials {
03774
              enum hermite_kind {
    probabilist,
03776
03778
                  physicist
03779
03780
          }
03781
          // hermite
03782
03783
          namespace internal {
03784
              template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03785
              struct hermite_helper {};
03786
03787
              template<size_t deg, typename I>
03788
              struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
03789
               private:
03790
                  using hnm1 = typename hermite_helper<deg - 1,
      known_polynomials::hermite_kind::probabilist, I>::type;
03791
                  using hnm2 = typename hermite_helper<deg - 2,
      known_polynomials::hermite_kind::probabilist, I>::type;
03792
03793
               public:
03794
                  using type = typename polynomial<I>::template sub_t<
                       typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnm1>,
03795
03796
                       typename polynomial<I>::template mul_t<
03797
                           typename polynomial<I>::template inject_constant_t<deg - 1>,
03798
                          hnm2
03799
03800
                  >;
03801
              } ;
03802
03803
              template<size_t deg, typename I>
03804
              struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
               private:
03805
                  using hnml = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
03806
      I>::type;
03807
                  using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
     I>::type;
03808
03809
               public:
03810
                  using type = typename polynomial<I>::template sub_t<
03811
                       // 2X Hn-1
03812
                       typename polynomial<I>::template mul_t<</pre>
03813
                           typename pi64::val<typename I::template inject_constant_t<2>,
03814
                           typename I::zero>, hnm1>,
03815
03816
                       typename polynomial<I>::template mul_t<
                           typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03817
03818
03819
03820
                  >;
03821
              };
03822
03823
              template<typename I>
03824
              struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03825
                  using type = typename polynomial<I>::one;
03826
              };
03827
03828
              template<typename I>
03829
              struct hermite helper<1, known polynomials::hermite kind::probabilist, I> {
```

```
using type = typename polynomial<I>::X;
03831
03832
03833
              template<typename I>
03834
              struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03835
                  using type = typename pi64::one;
03837
03838
              template<typename I>
03839
              struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03840
                  // 2X
                  using type = typename polynomial<I>::template val<</pre>
03841
03842
                      typename I::template inject_constant_t<2>,
03843
                      typename I::zero>;
03844
03845
          } // namespace internal
03846
03847
          // legendre
03848
          namespace internal {
03849
              template<size_t n, typename I>
03850
              struct legendre_helper {
               private:
03851
                  using O = FractionField<I>;
03852
                  using PQ = polynomial<Q>;
03853
                  // 1/n constant
// (2n-1)/n X
03854
03855
03856
                  using fact_left = typename PQ::template monomial_t<
03857
                      makefraction_t<I,</pre>
03858
                          typename I::template inject_constant_t<2*n-1>,
03859
                          typename I::template inject_constant_t<n>
03860
                      >,
03861
                  1>;
03862
                  // (n-1) / n
03863
                  using fact_right = typename PQ::template val<
                      makefraction_t<I,
03864
                          typename I::template inject_constant_t<n-1>,
03865
03866
                          typename I::template inject_constant_t<n>>;
03867
03868
03869
                  using type = PQ::template sub_t<
03870
                           typename PQ::template mul_t<
03871
                               fact_left,
03872
                               typename legendre_helper<n-1, I>::type
03873
03874
                           typename PQ::template mul_t<
03875
                               fact_right,
03876
                               typename legendre_helper<n-2, I>::type
03877
03878
                      >;
03879
              };
03880
03881
              template<typename I>
03882
              struct legendre_helper<0, I> {
03883
                 using type = typename polynomial<FractionField<I>::one;
03884
03885
03886
              template<typename I>
03887
              struct legendre_helper<1, I> {
03888
                 using type = typename polynomial<FractionField<I>::X;
03889
          } // namespace internal
03890
03891
03892
          // bernoulli polynomials
03893
          namespace internal {
03894
              template<size_t n>
03895
              struct bernoulli_coeff {
03896
                  template<typename T, size_t i>
03897
                  struct inner {
03898
                   private:
03899
                      using F = FractionField<T>;
03900
                   public:
03901
                      using type = typename F::template mul_t<</pre>
                           typename F::template inject_ring_t<combination_t<T, i, n»,
03902
03903
                           bernoulli_t<T, n-i>
03904
                      >;
03905
                  };
03906
              };
03907
          } // namespace internal
03908
03909
          namespace internal {
03910
             template<size t n>
03911
              struct touchard_coeff {
03912
                  template<typename T, size_t i>
03913
                  struct inner {
03914
                      using type = stirling_2_t<T, n, i>;
03915
                  } ;
03916
              };
```

```
} // namespace internal
03918
03919
          namespace internal {
03920
              template<typename I = aerobus::i64>
03921
              struct AbelHelper {
03922
               private:
03923
                  using P = aerobus::polynomial<I>;
03924
03925
               public:
03926
                  // to keep recursion working, we need to operate on a \star n and not just a
03927
                  template<size_t deg, I::inner_type an>
03928
                  struct Inner {
                       // abel(n, a) = (x-an) * abel(n-1, a)
using type = typename aerobus::mul_t
03929
03930
03931
                           typename Inner<deg-1, an>::type,
03932
                           typename aerobus::sub_t<typename P::X, typename P::template inject_constant_t<an>>
03933
                       >;
03934
                  };
03935
03936
                   // abel(0, a) = 1
03937
                   template<I::inner_type an>
03938
                   struct Inner<0, an>
03939
                       using type = P::one;
03940
03941
03942
                   // abel(1, a) = X
                   template<I::inner_type an>
03943
                   struct Inner<1, an> {
03944
03945
                       using type = P::X;
03946
                   };
03947
          };
} // namespace internal
03948
03949
03951
          namespace known_polynomials {
03952
03961
              template<size_t n, auto a, typename I = aerobus::i64>
03962
              using abel = typename internal::AbelHelper<I>::template Inner<n, a*n>::type;
03963
03971
              template <size_t deg, typename I = aerobus::i64>
03972
              using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
03973
03983
              template <size_t deg, typename I = aerobus::i64>
03984
              using chebyshev U = typename internal::chebyshev helper<2, deg, I>::type;
03985
03995
              template <size_t deg, typename I = aerobus::i64>
03996
              using laguerre = typename internal::laguerre_helper<deg, I>::type;
03997
              template <size_t deg, typename I = aerobus::i64>
04004
              using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
04005
      I>::tvpe;
04006
04013
              template <size_t deg, typename I = aerobus::i64>
04014
              using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
04015
04026
              template<size_t i, size_t m, typename I = aerobus::i64>
04027
              using bernstein = typename internal::bernstein helper<i, m, I>::type;
04028
04038
              template<size_t deg, typename I = aerobus::i64>
04039
              using legendre = typename internal::legendre_helper<deg, I>::type;
04040
04050
              template<size_t deg, typename I = aerobus::i64>
04051
              using bernoulli = taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
04052
04059
              template<size_t deg, typename I = aerobus::i64>
04060
              using allone = typename internal::AllOneHelper<deg, I>::type;
04061
04069
              template<size_t deg, typename I = aerobus::i64>
04070
              using bessel = typename internal::BesselHelper<deg, I>::type;
04071
04079
              template<size_t deg, typename I = aerobus::i64>
04080
              using touchard = taylor<I, internal::touchard_coeff<deg>::template inner, deg>;
04081
             // namespace known_polynomials
04082 } // namespace aerobus
04083
04084
04085 #ifdef AEROBUS_CONWAY_IMPORTS
04086
04087 // conway polynomials
04088 namespace aerobus {
          template<int p, int n>
struct ConwayPolynomial {};
04092
04093
04095 #ifndef DO_NOT_DOCUMENT
04096
          #define ZPZV ZPZ::template val
04097
          #define POLYV aerobus::polynomial<ZPZ>::template val
      template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>, // NOLINT
04098
```

```
04099
                                                                      template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
                                         POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
 04100
                                                                 template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
  04101
                                                                    template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
  04103
                                                               template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
                                                                  template<> struct ConwayPolynomial<2, 7> { using ZPZ = aerobus::zpz<2>; using type =
 04104
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
 04105
                                                                    template<> struct ConwayPolynomial<2, 8> { using ZPZ = aerobus::zpz<2>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV×1»; }; // NOLINT
                                                               template<> struct ConwayPolynomial<2, 9> { using ZPZ = aerobus::zpz<2>; using type =
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; //
                                          NOLINT
                                         template<> struct ConwayPolynomial<2, 10> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<
 04107
                                          ZPZV<1»: }: // NOLINT</pre>
                                         template<> struct ConwayPolynomial<2, 11> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                         ZPZV<0>, ZPZV<1»; }; // NOLINT
    template<> struct ConwayPolynomial<2, 12> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
 04109
                                                                 template<> struct ConwayPolynomial<2, 13> { using ZPZ = aerobus::zpz<2>; using type =
 04110
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
                                         template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
04111
                                         ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2, ZPZV<1>, ZPZV<2, ZPZV<1>, ZPZV<2, ZPZV<1>, ZPZV<2, ZPZV<1, ZPZV
 04112
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZ
 04113
                                                                      template<> struct ConwayPolynomial<2, 17> { using ZPZ = aerobus::zpz<2>; using type
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          template<> struct ConwayPolynomial<2, 18> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
 04115
                                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1»; };</pre>
                                         template<> struct ConwayPolynomial<2, 19> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             7.PZV<0>
                                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
                                          NOLINT
 04117
                                         template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
                                           ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1</pre>; };
 04118
                                                                   template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
                                       POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
                                                                  template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
 04119
                                         POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2»; }; // NOLINT
                                                                      template<> struct ConwayPolynomial<3, 3> { using ZPZ = aerobus::zpz<3>; using type =
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT template<> struct ConwayPolynomial<3, 4> { using ZPZ = aerobus::zpz<3>; using type =
                                         POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; }; // NOLINT
                                                                 template<> struct ConwayPolynomial<3, 5> { using ZPZ = aerobus::zpz<3>; using type =
 04122
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT
  04123
                                                                    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
 04124
                                                                 template<> struct ConwayPolynomial<3, 7> { using ZPZ = aerobus::zpz<3>; using type =
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1»; }; // NOLINT
  04125
                                                                    \texttt{template<>} \texttt{struct ConwayPolynomial<3, 8> \{ \texttt{using ZPZ = aerobus::zpz<3>; using type = ae
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2
                                                                  template<> struct ConwayPolynomial<3, 9> { using ZPZ = aerobus::zpz<3>; using type =
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; //
 04127
                                                               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>
                                         template<> struct ConwayPolynomial<3, 11> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
 04128
                                          ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
 04129
                                                                  template<> struct ConwayPolynomial<3, 12> { using ZPZ = aerobus::zpz<3>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<2
                                                                    template<> struct ConwayPolynomial<3, 13> { using ZPZ = aerobus::zpz<3>; using type
 04130
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT</pre>
  04131
                                                                 template<> struct ConwayPolynomial<3, 14> { using ZPZ = aerobus::zpz<3>; using type =
                                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<3, 15> { using ZPZ = aerobus::zpz<3>; using type =
  04132
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<0>,
                           ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<3, 16> { using ZPZ = aerobus::zpz<3>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2»; }; // NOLINT
   template<> struct ConwayPolynomial<3, 17> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04134
                           template<> struct ConwayPolynomial<3, 18> { using ZPZ = aerobus::zpz<3>; using type =
04135
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>; // NOLINT template<> struct ConwayPolynomial<3, 19> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZ
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; //</pre>
                          template<> struct ConwayPolynomial<3, 20> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; };
04137
                                            template<> struct ConwayPolynomial<5, 1> { using ZPZ = aerobus::zpz<5>; using type =
                          POLYV<ZPZV<1>, ZPZV<3»; }; // NOLINT
04139
                                            template<> struct ConwayPolynomial<5, 2> { using ZPZ = aerobus::zpz<5>; using type =
                          POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<5, 3> { using ZPZ = aerobus::zpz<5>; using type =
04140
                         POLYV-ZPZV-1>, ZPZV-0>, ZPZV-3>, ZPZV-3>; ); // NOLINT template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
04141
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
04142
                                           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
04143
                                           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
04144
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3», ZPZV<3», // NOLINT
                                         template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
04145
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
04146
                                            template<> struct ConwayPolynomial<5, 9> { using ZPZ = aerobus::zpz<5>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<3»; }; //
04147
                                             template<> struct ConwayPolynomial<5, 10> { using ZPZ = aerobus::zpz<5>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5>, ZPZV<5-, ZPZV<5
                           ZPZV<2»; }; // NOLINT</pre>
                          \label{eq:convayPolynomial} $$$ template<> struct ConwayPolynomial<5, 11> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04148
                           ZPZV<3>, ZPZV<3»; }; // NOLINT
                          template<> struct ConwayPolynomial<5, 12> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>,
                           ZPZV<3>, ZPZV<2>, ZPZV<2»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<5, 13> { using ZPZ = aerobus::zpz<5>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<5, 14> { using ZPZ = aerobus::zpz<5>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4>,
                           ZPZV<2>, ZPZV<3>, ZPZV<0>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial5, 15> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZP
04152
                          ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4»; }; // NOLINT
template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type =
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>,
                           ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
04154
                                              template<> struct ConwayPolynomial<5, 17> { using ZPZ = aerobus::zpz<5>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            template<> struct ConwayPolynomial<5, 18> { using ZPZ = aerobus::zpz<5>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>,
                           ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<2»; };</pre>
                          template<> struct ConwayPolynomial<5, 19> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
04156
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3»; }; //</pre>
                                             template<> struct ConwayPolynomial<5, 20> { using ZPZ = aerobus::zpz<5>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3
                           ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<0>, ZPZV<1>, ZPZV<2»; };</pre>
                           // NOLINT
04158
                                           template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
                          POLYV<ZPZV<1>, ZPZV<4»; }; // NOLINT
                                             template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
                          POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3»; }; // NOLINT
04160
                                            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 =
04161
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; };
                                                                                                                                                                                                                                                                              // NOLINT
                                            template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
04162
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT
04163
                                           template<> struct ConwayPolynomial<7, 6> { using ZPZ = aerobus::zpz<7>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<4>, ZPZV<6>, ZPZV<6>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<7, 7> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<4»; }; // NOLINT
04164
```

```
template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
04165
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<3>; }; // NOLINT template<> struct ConwayPolynomial<7, 9> { using ZPZ = aerobus::zpz<7>; using type =
 04166
                                   NOLINT
                                                        template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type =
 04167
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<3>,
                                   ZPZV<3»; }; // NOLINT</pre>
                                                      template<> struct ConwayPolynomial<7, 11> { using ZPZ = aerobus::zpz<7>; using type =
 04168
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                      template<> struct ConwayPolynomial<7, 12> { using ZPZ = aerobus::zpz<7>; using type
 04169
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<5>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<0>, ZPZV<5>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<5>, ZPZV<5-, ZPZV<5
                                                     template<> struct ConwayPolynomial<7, 13> { using ZPZ = aerobus::zpz<7>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  template<> struct ConwayPolynomial
t
tonwayPolynomial
t
t
polyv<2pzv<1>, ZPZV<0>, ZPZV<0 , ZPZ
 04171
                                   ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<3»; }; // NOLINT</pre>
                                  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<3>, ZPZV<4>, ZPZV<1>, ZPZV<6>, ZPZV<2>, ZPZV<4>, ZPZV<3»; }; // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<7, 17> { using ZPZ = aerobus::zpz<7>; using type =
 04174
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   04175
                                  template<> struct ConwayPolynomial<7, 18> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<6>, ZPZV<1>,
                                   ZPZV<6>, ZPZV<5>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<2>, ZPZV<3»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  // NOLINT
                                                        template<> struct ConwayPolynomial<7, 19> { using ZPZ = aerobus::zpz<7>; using type
 04176
                                   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<4»; }; //</pre>
                                   NOLINT
                                  template<> struct ConwayPolynomial<7, 20> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>,
 04177
                                    ZPZV<2>, ZPZV<5>, ZPZV<2>, ZPZV<3>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3</pre>
                                    // NOLINT
                                                       template<> struct ConwayPolynomial<11, 1> { using ZPZ = aerobus::zpz<11>; using type =
                                  POLYV<ZPZV<1>, ZPZV<9»; }; // NOLINT
                                                        template<> struct ConwayPolynomial<11, 2> { using ZPZ = aerobus::zpz<11>; using type =
 04179
                                 POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2»; }; // NOLINT
                                                           template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<9»; }; // NOLINT template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type =
  04181
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT

template<> struct ConwayPolynomial<11, 5> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<0>, ZPZV<9»; }; // NOLINT
  04182
 04183
                                                         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
  04184
                                                       template<> struct ConwayPolynomial<11, 7> { using ZPZ = aerobus::zpz<11>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<9»; }; // NOLINT
 04185
                                                       template<> struct ConwayPolynomial<11, 8> { using ZPZ = aerobus::zpz<11>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<1>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<11, 9> { using ZPZ = aerobus::zpz<11>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9»; }; //
 04187
                                                           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>
                                                         template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                     // NOLINT
                                    ZPZV<10>, ZPZV<9»; };</pre>
                                                     template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type =
 04189
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZPZV<6>, ZPZV<6 , ZPZV<6
                                                       template<> struct ConwayPolynomial<11, 13> { using ZPZ = aerobus::zpz<11>; using type =
                                  POLYY<ZPZV<0>, ZPZV<0>, ZPZV<0
                                                      template<> struct ConwayPolynomial<11, 14> { using ZPZ = aerobus::zpz<11>; using type =
 04191
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
 04192
                                   ZPZV<7>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<0>, ZPZV<9»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<11, 16> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1 ,
 04193
 04194
                                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<9»; }; // NOLINT</pre>
  04195
                                                      template<> struct ConwayPolynomial<11, 18> { using ZPZ = aerobus::zpz<11>; using type
                                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
  04196
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>
                          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<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<5>, ZPZV<5</pre>
04197
 04198
                                            template<> struct ConwayPolynomial<13, 1> { using ZPZ = aerobus::zpz<13>; using type
                         POLYV<ZPZV<1>, ZPZV<11»; }; // NOLINT
04199
                                          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 =
 04200
                         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
 04202
                                           template<> struct ConwayPolynomial<13, 5> { using ZPZ = aerobus::zpz<13>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<11»; }; // NOLINT
                                           template<> struct ConwayPolynomial<13, 6> { using ZPZ = aerobus::zpz<13>; using type =
04203
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<11>, ZPZV<11>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<13, 7> { using ZPZ = aerobus::zpz<13>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<11»; }; //
 04205
                                           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 =
04206
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<
                          // NOLINT
                                         template<> struct ConwayPolynomial<13, 10> { using ZPZ = aerobus::zpz<13>; using type =
04207
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>,
                          ZPZV<2»; }; // NOLINT</pre>
04208
                                         template<> struct ConwayPolynomial<13, 11> { using ZPZ = aerobus::zpz<13>; using type =
                           \texttt{POLYV} < 2 \texttt{PZV} < 1 >, \ \texttt{ZPZV} < 0 >, \ \texttt{ZPZV} 
                          ZPZV<3>, ZPZV<11»; };</pre>
                                                                                                                           // NOLINT
                                           template<> struct ConwayPolynomial<13, 12> { using ZPZ = aerobus::zpz<13>; using type =
 04209
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<8>, ZPZV<11>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4 , ZPZV<
                                            template<> struct ConwayPolynomial<13, 13> { using ZPZ = aerobus::zpz<13>; using type
04210
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            template<> struct ConwayPolynomial<13, 14> { using ZPZ = aerobus::zpz<13>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<6>,
                          ZPZV<11>, ZPZV<7>, ZPZV<10>, ZPZV<10>, ZPZV<2»; }; // NOLINT
                         template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<12>,
04212
                         ZPZV<2>, ZPZV<11>, ZPZV<10>, ZPZV<11>, ZPZV<8>, ZPZV<11», ZPZV<8, ZPZV<13; // NOLINT template<> struct ConwayPolynomial<13, 16> { using ZPZ = aerobus::zpz<13>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<8>, ZPZV<2>, ZPZV<12>, ZPZV<9>, ZPZV<12>, ZPZV<6>, ZPZV<2»; }; // NOLINT
    template<> struct ConwayPolynomial<13, 17> { using ZPZ = aerobus::zpz<13>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<6>, ZPZV<11»; }; // NOLINT
                                            template<> struct ConwayPolynomial<13, 18> { using ZPZ = aerobus::zpz<13>; using type
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10, ZPZV<4>, ZPZV<11>,
                           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>, ZPZ
04216
                           ZPZV<0>, ZPZV<0</pre>
                                           template<> struct ConwayPolynomial<13, 20> { using ZPZ = aerobus::zpz<13>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<10>, Z
                           ZPZV<9>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<7>, ZPZV<4>, ZPZV<0>, ZPZV<4>, ZPZV<8>, ZPZV<11>, ZPZV<2»; };</pre>
                           // NOLINT
04218
                                          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 =
                         POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»; }; // NOLINT
                                          template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
 04220
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
04221
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<10>, ZPZV<3»; }; // NOLINT
                         template<> struct ConwayPolynomial<17, 5> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT
 04222
 04223
                                         template<> struct ConwayPolynomial<17, 6> { using ZPZ = aerobus::zpz<17>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>; ); // NOLINT template<> struct ConwayPolynomial<17, 7> { using ZPZ = aerobus::zpz<17>; using type =
 04224
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; // NOLINT template<> struct ConwayPolynomial<17, 8> { using ZPZ = aerobus::zpz<17>; using type =
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<12>, ZPZV<0>, ZPZV<6>, ZPZV<3»; };
 04226
                                           template<> struct ConwayPolynomial<17, 9> { using ZPZ = aerobus::zpz<17>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<14»; };
                           // NOLINT
                         template<> struct ConwayPolynomial<17, 10> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<6>, ZPZV<5>, ZPZV<9>, ZPZV<12>,
04227
                          ZPZV<3»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<17, 11> { using ZPZ = aerobus::zpz<17>; using type
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<14»; }; // NOLINT template<> struct ConwayPolynomial<17, 12> { using ZPZ = aerobus::zpz<17>; using type =
04229
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<13>, ZPZV<6>,
                                ZPZV<14>, ZPZV<9>, ZPZV<3»; }; // NOLINT</pre>
                                                  template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<14»; }; // NOLINT
template<> struct ConwayPolynomial<17, 14> { using ZPZ = aerobus::zpz<17>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<8>,
                                 ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::zpz<17>; using type =
04232
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<16>, ZPZV<16>, ZPZV<14>, ZPZV<14>, ZPZV<14»; }; // NOLINT template<> struct ConwayPolynomial<17, 16> { using ZPZ = aerobus::zpz<17>; using type =
04233
                               POLYYCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<3, ZPZV<1>, ZPZV<3, ZPZV<3
                                                 template<> struct ConwayPolynomial<17, 17> { 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<16>, ZPZV<14»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<17, 18> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
04235
                                ZPZV<7>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<11>, ZPZV<13>, ZPZV<13>, ZPZV<9>, ZPZV<3»; };</pre>
                               template<> struct ConwayPolynomial<17, 19> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, 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<5>,
                                 ZPZV<16>, ZPZV<14>, ZPZV<13>, ZPZV<3>, ZPZV<14>, ZPZV<9>, ZPZV<1>, ZPZV<13>, ZPZV<2>, ZPZV<5>,
                                 ZPZV<3»; }; // NOLINT</pre>
04238
                                                      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 =
04239
                               POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<19, 3> { using ZPZ = aerobus::zpz<19>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type =
04241
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<19, 5> { using ZPZ = aerobus::zpz<19>; using type =
04242
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17»; }; // NOLINT
04243
                                                      template<> struct ConwayPolynomial<19, 6> { using ZPZ = aerobus::zpz<19>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<6>, ZPZV<6>, ZPZV<27>, ZPZV<6>, ZPZV<29; }; // NOLINT template<> struct ConwayPolynomial<19, 7> { using ZPZ = aerobus::zpz<19>; using type =
04244
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<1>; ); // NOLINT template<> struct ConwayPolynomial<19, 8> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<10>, ZPZV<3>, ZPZV<2»; };
04245
                               template<> struct ConwayPolynomial<19, 9> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<14>, ZPZV<16>, ZPZV<16>, ZPZV<17»; };
                                // NOLINT
                               template<> struct ConwayPolynomial<19, 10> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<13>, ZPZV<17>, ZPZV<3>, ZPZV<4>,
04247
                                ZPZV<2»: }: // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<19, 11> { using ZPZ = aerobus::zpz<19>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<8>, ZPZV<17»; }; // NOLINT</pre>
04249
                                                     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
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<17»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<19, 14> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<11>, ZPZV<11>, ZPZV<15>, ZPZV<16>, ZPZV<7>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<19, 15> { using ZPZ = aerobus::zpz<19>; using type =
04251
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZ
                                 ZPZV<11>, ZPZV<13>, ZPZV<15>, ZPZV<14>, ZPZV<0>, ZPZV<17»; }; // NOLINT</pre>
04253
                                                  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<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<2>, ZPZV<2>; // NOLINT template<> struct ConwayPolynomial<19, 17> { using ZPZ = aerobus::zpz<19>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<17»; };</pre>
04255
                                                 template<> struct ConwayPolynomial<19, 18> { using ZPZ = aerobus::zpz<19>; using type =
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<7>, ZPZV<17>, ZPZV<5>, ZPZV<6>, ZPZV<16>, ZPZV<5>, ZPZV<7>, ZPZV<14>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 19> { using ZPZ = aerobus::zpz<19>; using type =
04256
                                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<18>, ZPZV<17»; };</pre>
                               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<7>, ZPZV<8>, ZPZV<6>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<6>, ZPZV<11>, ZPZV<2»;</pre>
                                }; // NOLINT
                                                      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»; };
                       template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type =
04262
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<18»; }; // NOLINT
04263
                                       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
04264
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<18»; }; // NOLINT
04265
                                     template<> struct ConwayPolynomial<23, 8> { using ZPZ = aerobus::zpz<23>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5»; }; // NOLINT
                                    template<> struct ConwayPolynomial<23, 9> { using ZPZ = aerobus::zpz<23>; using type =
04266
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<9>, ZPZV<18»; };
                                     template<> struct ConwayPolynomial<23, 10> { using ZPZ = aerobus::zpz<23>; using type
04267
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<5>, ZPZV<15>, ZPZV<6>, ZPZV<1>, ZPZV<5»; }; // NOLINT
                       template<> struct ConwayPolynomial<23, 11> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>,
04268
                       ZPZV<7>, ZPZV<18»; };</pre>
                                                                                                                  // 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<12>,
                       ZPZV<18>, ZPZV<12>, ZPZV<5»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<23, 13> { using ZPZ = aerobus::zpz<23>; using type =
04270
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<18»; };</pre>
                                                                                                                                                                                        // NOLINT
                                     template<> struct ConwayPolynomial<23, 14> { using ZPZ = aerobus::zpz<23>; using type =
04271
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<1>,
                       ZPZV<18>, ZPZV<19>, ZPZV<1>, ZPZV<22>, ZPZV<5»; }; // NOLINT
   template<> struct ConwayPolynomial<23, 15> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
04272
                       ZPZV-8>, ZPZV-15>, ZPZV-9>, ZPZV-7>, ZPZV-18>, ZPZV-18>; ]; // NOLINT template<> struct ConwayPolynomial<23, 16> { using ZPZ = aerobus::zpz<23>; using type =
04273
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<19>, ZPZV<16>, ZPZV<13>, ZPZV<1>, ZPZV<14>, ZPZV<17>, ZPZV<5>; // NOLINT
template<> struct ConwayPolynomial<23, 17> { using ZPZ = aerobus::zpz<23>; using type =
04274
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<18»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<23, 18> { using ZPZ = aerobus::zpz<23>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, 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
template<> struct ConwayPolynomial<23, 19> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZ
04276
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5</pre>
                       NOLINT
04277
                                       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 =
04278
                      POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; }; // NOLINT
                                       template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
                     POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<27»; }; // NOLINT template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
                       template<> struct ConwayPolynomial<29, 5> { using ZPZ = aerobus::zpz<29>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<27»; }; // NOLINT</pre>
04281
                                       template<> struct ConwayPolynomial<29, 6> { using ZPZ = aerobus::zpz<29>; using type =
04282
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<17>, ZPZV<13>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type
04283
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<27»; }; // NOLINT template<> struct ConwayPolynomial<29, 8> { using ZPZ = aerobus::zpz<29>; using type =
04284
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<23>, ZPZV<2); }; //
                       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<4>, ZPZV<22>, ZPZV<22>, ZPZV<27»; };
04286
                                     template<> struct ConwayPolynomial<29, 10> { using ZPZ = aerobus::zpz<29>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2
                       ZPZV<2»: }: // NOLINT
                                      template<> struct ConwayPolynomial<29, 11> { using ZPZ = aerobus::zpz<29>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<8>, ZPZV<27»; };</pre>
                                                                                                                // NOLINT
                                    template<> struct ConwayPolynomial<29, 12> { using ZPZ = aerobus::zpz<29>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<28>, ZPZV<9>, ZPZV<16>, ZPZV<25>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>; }; // NOLINT
                                       template<> struct ConwayPolynomial<29, 13> { using ZPZ = aerobus::zpz<29>; using type
04289
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<27»; };</pre>
                                                                                                                                                                                        // NOLINT
04290
                                      template<> struct ConwayPolynomial<29, 14> { using ZPZ = aerobus::zpz<29>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<3>, ZPZV<14>, ZPZV<14>, ZPZV<10>,
                       ZPZV<21>, ZPZV<18>, ZPZV<27>, ZPZV<5>, ZPZV<2»; ); // NOLINT
  template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::zpz<29>; using type
04291
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>,
                       ZPZV<14>, ZPZV<8>, ZPZV<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
                       ZPZV<2>, ZPZV<18>, ZPZV<23>, ZPZV<1>, ZPZV<27>, ZPZV<10>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<29, 17> { using ZPZ = aerobus::zpz<29>; using type =
04293
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                            template<> struct ConwayPolynomial<29, 18> { using ZPZ = aerobus::zpz<29>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<27»; };</pre>
                                             template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<28»; }; // NOLINT
                                             template<> struct ConwayPolynomial<31, 2> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<29>, ZPZV<3»; }; // NOLINT
                                               template<> struct ConwayPolynomial<31, 3> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; // NOLINT template<> struct ConwayPolynomial<31, 4> { using ZPZ = aerobus::zpz<31>; using type =
 04299
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<31, 5> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; }; // NOLINT
04300
                                              template<> struct ConwayPolynomial<31, 6> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<16>, ZPZV<8>, ZPZV<8>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type =
 04302
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2+>, ZPZV<28»; }; // NOLINT
                           template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<12>, ZPZV<24>, ZPZV<3»; };</pre>
04303
                                            template<> struct ConwayPolynomial<31, 9> { using ZPZ = aerobus::zpz<31>; using type =
04304
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<20>, ZPZV<29>, ZPZV<28»; };
                            // NOLINT
                                             template<> struct ConwayPolynomial<31, 10> { using ZPZ = aerobus::zpz<31>; using type =
04305
                            POLYV<ZPZV<1>, 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 =
 04306
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<20>, ZPZV<28»; }; // NOLINT</pre>
04307
                                              template<> struct ConwayPolynomial<31, 12> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<14>, ZPZV<28>, ZPZV<2>, ZPZV<9>, ZPZV<25>, ZPZV<25>, ZPZV<12>, ZPZV<3»; }; // NOLINT
                                               template<> struct ConwayPolynomial<31, 13> { using ZPZ = aerobus::zpz<31>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<28»; }; // NOLINT
                           ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<20»; ;; // NOLINI
    template<> struct ConwayPolynomial<31, 14> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
04309
                            ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<6>, ZPZV<3»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<29>, ZPZV<12>, ZPZV<13>, ZPZV<23>, ZPZV<25>, ZPZV<28*; }; // NOLINT template<> struct ConwayPolynomial<31, 16> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3*; }; // NOLINT template<> struct ConwayPolynomial<31, 17> { using ZPZ = aerobus::zpz<31>; using type = POLYV<2PZV<1 > ZPZV<26 > ZPZV<28 > ZPZV<28 > ZPZV<11 > ZPZV<19 > ZPZV<27 > ZPZV<3 > ZP
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<28»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                                           // 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<0>, ZPZV<2>, ZPZV<27>, ZPZV<27>, ZPZV<25>, ZPZV<24>, ZPZV<25>, ZPZV<26>, ZPZV<3»; }; // NOLINT
04313
                                               template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<28»; }; //</pre>
                            NOLINT
04315
                                            template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
04316
                                               template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2»; }; // NOLINT
04317
                                            template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT
template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<37, 5> { using ZPZ = aerobus::zpz<37>; using type =
 04318
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35»; }; // NOLINT
                                              template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
 04320
                           template<> struct ConwayPolynomial<37, 7> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<35»; }; // NOLINT</pre>
 04321
                                              template<> struct ConwayPolynomial<37, 8> { using ZPZ = aerobus::zpz<37>; using type =
04322
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<20>, ZPZV<27>, ZPZV<27>, ZPZV<1>, ZPZV<28; }; // NOLINT
                                            template<> struct ConwayPolynomial<37, 9> { using ZPZ = aerobus::zpz<37>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<20, ZPZV<32>, ZPZV<35»; };
                            // NOLINT
04324
                                            template<> struct ConwayPolynomial<37, 10> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<11>, ZPZV<4>,
                            ZPZV<2»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<37, 11> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                            ZPZV<2>, ZPZV<35»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<10>, ZPZV<23>, ZPZV<23>, ZPZV<23>, ZPZV<23>, ZPZV<24>, ZPZV<31>, ZPZV<31>
```

```
ZPZV<18>, ZPZV<33>, ZPZV<2»; };</pre>
                                        template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<37, 14> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<35>, ZPZV<35>, ZPZV<1>, ZPZV<32>, ZPZ
04328
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<31>,
                        ZPZV<28>, ZPZV<27>, ZPZV<13>, ZPZV<34>, ZPZV<33>, ZPZV<35»; }; // NOLINT
    template<> struct ConwayPolynomial<37, 17> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04330
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35»; }; // 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<0>, ZPZV<1>, ZPZV<8>, ZPZV<8>, ZPZV<15>,
                        ZPZV<1>, ZPZV<22>, ZPZV<20>, ZPZV<12>, ZPZV<32>, ZPZV<414, 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<3>, ZPZV<3 , ZPZV<3
                                         template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type =
                        POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                       template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
04334
                        POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6»; }; // NOLINT
04335
                                        template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
                        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<35»; ); // NOLINT template<> struct ConwayPolynomial<41, 4> { using ZPZ = aerobus::zpz<41>; using type =
 04336
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<23>, ZPZV<6»; }; // NOLINT
                                        template<> struct ConwayPolynomial<41, 5> { using ZPZ = aerobus::zpz<41>; using type =
 04337
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<14>, ZPZV<35»; }; // NOLINT
                                       template<> struct ConwayPolynomial<41, 6> { using ZPZ = aerobus::zpz<41>; using type =
04338
                        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<5»; };
 04340
                                        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»; };
                                       template<> struct ConwayPolynomial<41, 9> { using ZPZ = aerobus::zpz<41>, using type
04341
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<35>, ZPZV<35»; };
                                       template<> struct ConwayPolynomial<41, 10> { using ZPZ = aerobus::zpz<41>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<8>, ZPZV<80, ZPZV<30>,
                        ZPZV<6»; }; // NOLINT
                        template<> struct ConwayPolynomial<41, 11> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04343
04344
                                        template<> struct ConwayPolynomial<41, 12> { using ZPZ = aerobus::zpz<41>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26>, ZPZV<24>, ZPZV<21>, ZPZV<27>, ZPZV<6»; }; // NOLINT
04345
                                       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 =
04346
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>,
                        ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6>; }; // NOLINT
    template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
04347
                         ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<41, 17> { using ZPZ = aerobus::zpz<41>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<3
04349
                        template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<35»; }; //</pre>
                        NOLINT
                                       template<> struct ConwayPolynomial<43, 1> { using ZPZ = aerobus::zpz<43>; using type =
04351
                        POLYV<ZPZV<1>, ZPZV<40»; }; // NOLINT
 04352
                                         template<> struct ConwayPolynomial<43, 2> { using ZPZ = aerobus::zpz<43>; using type =
                        POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3»; }; // NOLINT
 04353
                                      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 =
 04354
                       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 =
                      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 =
04357
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<7>, ZPZV<40»; }; // NOLINT
                                        template<> struct ConwayPolynomial<43, 8> { using ZPZ = aerobus::zpz<43>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<39>, ZPZV<20>, ZPZV<24>, ZPZV<3»; };
                        NOLINT
04359
                                      template<> struct ConwayPolynomial<43, 9> { using ZPZ = aerobus::zpz<43>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<12>, ZPZV<39>, ZPZV<1>, ZPZV<40»; };
                         // NOLINT
```

```
template<> struct ConwayPolynomial<43, 10> { using ZPZ = aerobus::zpz<43>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<26>, ZPZV<36>, ZPZV<36>, ZPZV<5>, ZPZV<27>, ZPZV<24>,
                                    ZPZV<3»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
 04361
                                   template<> struct ConwayPolynomial<43, 12> { using ZPZ = aerobus::zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3+, ZPZV<7>, 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 , ZPZV<0 ,
04363
                                    ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<40»; }; // NOLINT</pre>
                                                            template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type =
04364
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2+, ZPZV<38, 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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3</pre>
04365
                                   ZPZV<22>, ZPZV<42>, ZPZV<42>, ZPZV<42>, ZPZV<43>, ZPZV<4
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 36>, \ \texttt{ZPZV} < 40»; \ \}; \ \ // \ \texttt{NOLINT} 
04367
                                                            template<> struct ConwayPolynomial<43, 18> { using ZPZ = aerobus::zpz<43>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<28>, ZPZV<41>, ZPZV<24>, ZPZV<25 , ZPZV<25 , ZPZV<26>, ZPZV<26 , ZPZV<26 , ZPZV<27 , ZPZV<28 , ZPZV<38 ,
04368
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<40»; };</pre>
                                    NOLINT
04369
                                                           template<> struct ConwayPolynomial<47, 1> { using ZPZ = aerobus::zpz<47>; using type =
                                  POLYV<ZPZV<1>, ZPZV<42»; }; // NOLINT
                                                          template<> struct ConwayPolynomial<47, 2> { using ZPZ = aerobus::zpz<47>; using type =
                                  POLYV<ZPZV<1>, ZPZV<45>, ZPZV<5»; }; // NOLINT
                                                            template<> struct ConwayPolynomial<47, 3> { using ZPZ = aerobus::zpz<47>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42»; }; // NOLINT
                                                            template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type =
 04372
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; }; // NOLINT
                                  template<> struct ConwayPolynomial<47, 5> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>; }; // NOLINT
04373
                                                             template<> struct ConwayPolynomial<47, 6> { using ZPZ = aerobus::zpz<47>; using type =
 04374
                                   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 =
 04375
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<42»; }; // NOLINT 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»; };
 04376
                                   template<> struct ConwayPolynomial<47, 9> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<42»; };
                                    // NOLINT
                                   template<> struct ConwayPolynomial<47, 10> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42>, ZPZV<14>, ZPZV<14>, ZPZV<18>, ZPZV<45>, ZPZV<45>,
04378
                                    ZPZV<5»; }; // NOLINT
                                                            template<> struct ConwayPolynomial<47, 11> { using ZPZ = aerobus::zpz<47>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<6>, ZPZV<42»; }; // NOLINT
template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<46>, ZPZV<40>, ZPZV<35>, ZPZV<12>, ZPZV<46>,
04380
                                                             template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<42»; };</pre>
                                                                                                                                                                                                                                                                                        // NOLINT
                                   template<> struct ConwayPolynomial<47, 14> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<30>, ZPZV<30>, ZPZV<37>, ZPZV<37>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<39>, ZPZV<30>, 
04382
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     ZPZV<31>, ZPZV<14>, ZPZV<42>, ZPZV<13>, ZPZV<17>, ZPZV<42»; }; // NOLINT</pre>
04384
                                                        template<> struct ConwayPolynomial<47, 17> { using ZPZ = aerobus::zpz<47>; 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<6>, ZPZV<6>, ZPZV<41>, ZPZV<42>,
                                     ZPZV<26>, ZPZV<44>, ZPZV<24>, ZPZV<22>, ZPZV<11>, ZPZV<5>, ZPZV<45>, ZPZV<33>, ZPZV<5»; }; // NOLINT
04386
                                                        template<> struct ConwayPolynomial<47, 19> { using ZPZ = aerobus::zpz<47>; using type =
                                   POLYYCZPZV<1>, ZPZV<0>, ZPZV<0
                                    NOLINT
                                                            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 =
04389
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
 04390
                                  POLYVCZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<38>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<53, 5> { using ZPZ = aerobus::zpz<53>; using type =
 04391
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT
                                   template<> struct ConwayPolynomial<53, 6> { using ZPZ = aerobus::zpz<53>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<4>, ZPZV<45>, ZPZV<2»; }; // NOLINT</pre>
 04392
```

```
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<51»; }; // NOLINT
                           template<> struct ConwayPolynomial<53, 8> { using ZPZ = aerobus::zpz<53>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<1>, ZPZV<2»; };
                            template<> struct ConwayPolynomial<53, 9> { using ZPZ = aerobus::zpz<53>; using type
                 POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<51»; };
                             template<> struct ConwayPolynomial<53, 10> { using ZPZ = aerobus::zpz<53>; using type
04396
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<27>, ZPZV<25>, ZPZV<29>,
                 ZPZV<2»; }; // NOLINT</pre>
04397
                            template<> struct ConwayPolynomial<53, 11> { using ZPZ = aerobus::zpz<53>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                                                                      // NOLINT
                 ZPZV<15>, ZPZV<51»; };</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<41>, ZPZV<13>, ZPZV<10>, ZPZV<42>,
                  ZPZV<34>, ZPZV<41>, ZPZV<2»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<53, 13> { using ZPZ = aerobus::zpz<53>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                 ZPZV<0>, ZPZV<5>, ZPZV<28>, ZPZV<28>, ZPZV<51»; }; // NOLINT template<> struct ConwayPolynomial<53, 14> { using ZPZ = aerobus::zpz<53>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<4>, ZPZV<45>, ZPZV<45>, ZPZV<52>,
                 \text{ZPZV}<0>, \text{ZPZV}<37>, \text{ZPZV}<12>, \text{ZPZV}<23>, \text{ZPZV}<2»; }; // \text{NOLINT}
                 template<> struct ConwayPolynomial<53, 15> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>,
04401
                 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
                  template<> struct ConwayPolynomial<53, 18> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<52>, ZPZV<52>, ZPZV<52>, ZPZV<51>, ZPZV<51>, ZPZV<52>, ZPZV<52>, ZPZV<52>, ZPZV<52>, ZPZV<52>; // NOLINT
                             template<> struct ConwayPolynomial<53, 19> { using ZPZ = aerobus::zpz<53>; using type
04404
                 POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<51»; }; //</pre>
                 NOLINT
04405
                             template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
                 POLYV<ZPZV<1>, ZPZV<57»; }; // NOLINT
                             template<> struct ConwayPolynomial<59, 2> { using ZPZ = aerobus::zpz<59>; using type =
                 POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2»; }; // NOLINT
                            template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; }; // NOLINT template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
04408
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<59, 5> { using ZPZ = aerobus::zpz<59>; using type =
04409
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<57»; }; // NOLINT
04410
                             template<> struct ConwayPolynomial<59, 6> { using ZPZ = aerobus::zpz<59>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18>, ZPZV<38>, ZPZV<0>, ZPZV<2»; }; // NOLINT
                           template<> struct ConwayPolynomial<59, 7> { using ZPZ = aerobus::zpz<59>; using type =
04411
                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<57»; }; // NOLINT
                            template<> struct ConwayPolynomial<59, 8> { using ZPZ = aerobus::zpz<59>; using type =
                 POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<32>, ZPZV<2>, ZPZV<50>, ZPZV<2»; };
04413
                           template<> struct ConwayPolynomial<59, 9> { using ZPZ = aerobus::zpz<59>; using type
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<47>, ZPZV<57»; };
                 // NOLINT
                             template<> struct ConwayPolynomial<59, 10> { using ZPZ = aerobus::zpz<59>; using type
04414
                 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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                 ZPZV<6>, ZPZV<57»; }; // NOLINT</pre>
                 template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type = POLYV-ZPZV-1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<21>, ZPZV<38>, ZPZV<8>, ZPZV<1>, ZPZV<2»; }; // NOLINT
                           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 =
04418
                 POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<51>, ZPZV<11>,
                 ZPZV<13>, ZPZV<25>, ZPZV<32>, ZPZV<26>, ZPZV<2»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>>,
                 ZPZV<24>, ZPZV<23>, ZPZV<13>, ZPZV<39>, ZPZV<58>, ZPZV<57»; }; // NOLINT
    template<> struct ConwayPolynomial<59, 17> { using ZPZ = aerobus::zpz<59>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04420
                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59>; using type
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<2>
                 ZPZV<11>, ZPZV<14>, ZPZV<7>, ZPZV<44>, ZPZV<16>, ZPZV<47>, ZPZV<34>, ZPZV<32>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<59, 19> { using ZPZ = aerobus::zpz<59>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04422
                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<57»; }; //</pre>
04423
                            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 =
04424
                 POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; }; // NOLINT
```

```
04425
                                           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 =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<61, 5> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<59»; }; // NOLINT
template<> struct ConwayPolynomial<61, 6> { using ZPZ = aerobus::zpz<61>; using type =
04427
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<3>, ZPZV<29>, ZPZV<2»; };
04429
                                        template<> struct ConwayPolynomial<61, 7> { using ZPZ = aerobus::zpz<61>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<59»; };
04430
                                         template<> struct ConwayPolynomial<61, 8> { using ZPZ = aerobus::zpz<61>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>; template<> struct ConwayPolynomial<61, 9> { using ZPZ = aerobus::zpz<61>; using type =
04431
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<50>, ZPZV<50»; };
                           // NOLINT
                                          template<> struct ConwayPolynomial<61, 10> { using ZPZ = aerobus::zpz<61>; using type =
04432
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<15>, ZPZV<44>, ZPZV<16>, ZPZV<6>,
                          ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<61, 11> { using ZPZ = aerobus::zpz<61>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                          ZPZV<18>, ZPZV<59»; }; // NOLINT</pre>
04434
                                           template<> struct ConwayPolynomial<61, 12> { using ZPZ = aerobus::zpz<61>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<42>, ZPZV<33>, ZPZV<8>, ZPZV<38>, ZPZV<14>, ZPZV<15>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        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<26>, ZPZV<11>,
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<2»; ; // NOLINT template<> struct ConwayPolynomial<61, 15> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0>, ZPZV<0 , ZPZV<
                                        template<> struct ConwayPolynomial<61, 17> { using ZPZ = aerobus::zpz<61>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<59»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<61, 18> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<36>, ZPZV<13>,
04439
                          ZPZV<36>, ZPZV<4>, ZPZV<32>, ZPZV<57>, ZPZV<42>, ZPZV<25>, ZPZV<25>, ZPZV<52>, ZPZV<52>, ZPZV<50</pre>
                         template<> struct ConwayPolynomial<61, 19> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; };</pre>
                          NOLINT
                                           template<> struct ConwayPolynomial<67, 1> { using ZPZ = aerobus::zpz<67>; using type =
                         POLYV<ZPZV<1>, ZPZV<65»; }; // NOLINT
04442
                                           template<> struct ConwayPolynomial<67, 2> { using ZPZ = aerobus::zpz<67>; using type =
                         POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
04443
                                         template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65»; }; // NOLINT
                                          template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<54>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<67, 5> { using ZPZ = aerobus::zpz<67>; using type =
04445
                         04446
                                         template<> struct ConwayPolynomial<67, 6> { using ZPZ = aerobus::zpz<67>; using type =
                       POLYV-ZPZV-1>, ZPZV-0>, ZPZV-0>, ZPZV-63>, ZPZV-49>, ZPZV-5>, ZPZV-2>; ); // NOLINT template<> struct ConwayPolynomial<67, 7> { using ZPZ = aerobus::zpz<67>; using type
04447
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<65»; }; // NOLINT
                                        template<> struct ConwayPolynomial<67, 8> { using ZPZ = aerobus::zpz<67>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<46>, ZPZV<17>, ZPZV<64>, ZPZV<64>, ZPZV<2»; };
                          NOLINT
                                         template<> struct ConwayPolynomial<67, 9> { using ZPZ = aerobus::zpz<67>; using type =
04449
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<65>, ZPZV<65>, 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<15, ZPZV<18
                          ZPZV<2»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<67, 11> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>,
ZPZV<9>, ZPZV<65»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<67, 12> { using ZPZ = aerobus::zpz<67>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57>, ZPZV<27>, ZPZV<4>, ZPZV<64>, ZPZV<64>, ZPZV<64>, ZPZV<21>, ZPZV<27>, ZPZV<22>, ZPZV<21>, ZPZV<2
                         template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                         ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<65»; }; // NOLINT
template<> struct ConwayPolynomial<67, 14> { using ZPZ = aerobus::zpz<67>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
                         ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>; // NOLINT
template<> struct ConwayPolynomial<67, 17> { using ZPZ = aerobus::zpz<67>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0 , ZPZ
```

```
template<> struct ConwayPolynomial<67, 19> { using ZPZ = aerobus::zpz<67>; using type
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<0>
                                      NOLINT
 04459
                                                              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 =
                                     POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7»; }; // NOLINT
                                                            template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
 04461
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; }; // NOLINT template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
 04462
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<71, 5> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<64»; }; // NOLINT
 04463
 04464
                                                            template<> struct ConwayPolynomial<71, 6> { using ZPZ = aerobus::zpz<71>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<71, 7> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<6+; }; // NOLINT template<> struct ConwayPolynomial<71, 8> { using ZPZ = aerobus::zpz<71>; using type =
04465
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<22>, ZPZV<19>, ZPZV<7»; }; //
04467
                                                             template<> struct ConwayPolynomial<71, 9> { using ZPZ = aerobus::zpz<71>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43>, ZPZV<43>, ZPZV<64»; };
                                      // NOLINT
04468
                                                               template<> struct ConwayPolynomial<71, 10> { using ZPZ = aerobus::zpz<71>; using type =
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<17>, ZPZV<26>, ZPZV<14>, ZPZV<40>,
                                      ZPZV<7»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<71, 11> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                      ZPZV<48>, ZPZV<64»: }: // NOLINT</pre>
                                     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<28>, ZPZV<23>, ZPZV<3>; }; // NOLINT
                                                           template<> struct ConwayPolynomial<71, 13> { using ZPZ = aerobus::zpz<71>; using type =
                                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
04472
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<71, 17> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64*; }; // NOLINT
    template<> struct ConwayPolynomial<71, 19> { 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<0>, ZPZV<4>, ZPZV<4</pre>,  //
                                      NOLINT
04475
                                                             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 =
04476
                                     POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5»; }; // NOLINT
                                                               template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68»; }; // NOLINT
 04478
                                                             template<> struct ConwayPolynomial<73, 4> { using ZPZ = aerobus::zpz<73>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<73, 5> { using ZPZ = aerobus::zpz<73>; using type =
 04479
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<68»; }; // NOLINT
                                                                 template<> struct ConwayPolynomial<73, 6> { using ZPZ = aerobus::zpz<73>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<45>, ZPZV<23>, ZPZV<48>, ZPZV<5»; }; // NOLINT
                                                               template<> struct ConwayPolynomial<73, 7> { using ZPZ = aerobus::zpz<73>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<68»; };
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        // NOLINT
04482
                                                             template<> struct ConwayPolynomial<73, 8> { using ZPZ = aerobus::zpz<73>; using type =
                                      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<53>, ZPZV<53>, ZPZV<18>, ZPZV<18>, ZPZV<5»; }; //
                                     NOLINT
                                                             template<> struct ConwayPolynomial<73, 9> { using ZPZ = aerobus::zpz<73>; using type
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<72>, ZPZV<72>, ZPZV<68»; };
                                      // NOLINT
04484
                                     \label{eq:convayPolynomial} $$ $$ template<> struct ConwayPolynomial<73, 10> { using ZPZ = aerobus::zpz<73>; using type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<23>, ZPZV<33>, ZPZV<33>, ZPZV<32>, ZPZV<69>, Z
                                      ZPZV<5»; }; // NOLINT</pre>
                                                                 template<> struct ConwayPolynomial<73, 11> { using ZPZ = aerobus::zpz<73>; using type
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<5>, ZPZV<68»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<73, 12> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<66>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<46>, ZPZV<46 , ZPZV<47 , ZPZV<47
                                      ZPZV<29>, ZPZV<25>, ZPZV<5»; }; // NOLINT</pre>
                                                                 template<> struct ConwayPolynomial<73, 13> { using ZPZ = aerobus::zpz<73>; using type
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    POLYV<ZPZV<13, ZPZV<07, ZPZV<73, ZPZV<68»; }; // NOLINT

template<> struct ConwayPolynomial<73, 15> { using ZPZ = aerobus::zpz<73>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
                                                                 template<> struct ConwayPolynomial<73, 17> { 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<8>, ZPZV<68»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<73, 19> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
04490
```

```
NOLINT
                                    template<> struct ConwayPolynomial<79, 1> { using ZPZ = aerobus::zpz<79>; using type =
 04491
                     POLYV<ZPZV<1>, ZPZV<76»; }; // NOLINT
                                   template<> struct ConwayPolynomial<79, 2> { using ZPZ = aerobus::zpz<79>; using type =
 04492
                     POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3»; }; // NOLINT
                                    template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
04493
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76»; }; // NOLINT
 04494
                                    template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<79, 5> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<76»; }; // NOLINT
 04495
                                   template<> struct ConwayPolynomial<79, 6> { using ZPZ = aerobus::zpz<79>; using type =
04496
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<68>, ZPZV<3»; }; // NOLINT
                                     template<> struct ConwayPolynomial<79, 7> { using ZPZ = aerobus::zpz<79>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76»; };
 04498
                                  template<> struct ConwayPolynomial<79, 8> { using ZPZ = aerobus::zpz<79>; using type =
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<60>, ZPZV<59>, ZPZV<48>, ZPZV<3»; }; //
                     NOLINT
                                   template<> struct ConwayPolynomial<79, 9> { using ZPZ = aerobus::zpz<79>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<519>, ZPZV<76»; };
04500
                                   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<30>, ZPZV<42>,
                      ZPZV<3»; }; // NOLINT</pre>
04501
                                   template<> struct ConwayPolynomial<79, 11> { using ZPZ = aerobus::zpz<79>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<3>, ZPZV<76»; };</pre>
                                                                                                        // NOLINT
                     template<> struct ConwayPolynomial<79, 12> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<45>, ZPZV<52>, ZPZV<7>, ZPZV<40>,
                      ZPZV<59>, ZPZV<62>, ZPZV<3»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<78>, ZPZV<4>, ZPZV<76»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<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
04505
                       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<26»; };</pre>
04506
                                  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 =
 04507
                    POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81»; };
                                                                                                                                                                                             // NOLINT
 04509
                                  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 =
 04510
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<81»; }; // NOLINT
 04511
                                    template<> struct ConwayPolynomial<83, 6> { using ZPZ = aerobus::zpz<83>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<76>, ZPZV<32>, ZPZV<17>, ZPZV<2»; }; // NOLINT
 04512
                                  template<> struct ConwayPolynomial<83, 7> { using ZPZ = aerobus::zpz<83>; using type =
                     04513
                                  template<> struct ConwayPolynomial<83, 8> { using ZPZ = aerobus::zpz<83>; using type =
                      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<23>, ZPZV<42>, ZPZV<42»; };
                                  template<> struct ConwayPolynomial<83, 9> { using ZPZ = aerobus::zpz<83>; using type
 04514
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<24>, ZPZV<28>, ZPZV<81»; };
                      // NOLINT
04515
                                   template<> struct ConwayPolynomial<83, 10> { using ZPZ = aerobus::zpz<83>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5
                      ZPZV<2»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<17>, ZPZV<81»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<12>, ZPZV<31>, ZPZV<19>, ZPZV<65>,
ZPZV<55>, ZPZV<75>, ZPZV<75>, ZPZV<2»; }; // NOLINT</pre>
04517
                                     template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     template<> struct ConwayPolynomial<83, 17> { using ZPZ = aerobus::zpz<83>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04519
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<81*; }; // NOLINT
template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                            ZPZV<0>,
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<41>, ZPZV<81»; }; //</pre>
                      NOLINT
04521
                                   template<> struct ConwayPolynomial<89, 1> { using ZPZ = aerobus::zpz<89>; using type =
                     POLYV<ZPZV<1>, ZPZV<86»; }; // NOLINT
                                    template<> struct ConwayPolynomial<89, 2> { using ZPZ = aerobus::zpz<89>; using type =
                     POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; }; // NOLINT
 04523
                                  template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<86»; }; // NOLINT
template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; }; // NOLINT
 04524
```

```
04525
                                     template<> struct ConwayPolynomial<89, 5> { using ZPZ = aerobus::zpz<89>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<86»; }; // NOLINT
                                   template<> struct ConwayPolynomial<89, 6> { using ZPZ = aerobus::zpz<89>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<82>, ZPZV<80>, ZPZV<15>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6 >, ZPZ
04527
                                                                                                                                                                                                                                                                                                                                  // NOLTNT
                                   template<> struct ConwayPolynomial<89, 8> { using ZPZ = aerobus::zpz<89>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<40>, ZPZV<79>, ZPZV<3»; };
04529
                                  template<> struct ConwayPolynomial<89, 9> { using ZPZ = aerobus::zpz<89>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<12>, ZPZV<6>, ZPZV<86»; };
                      // NOLINT
                                    template<> struct ConwayPolynomial<89, 10> { using ZPZ = aerobus::zpz<89>; using type =
04530
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<33>, ZPZV<82>, ZPZV<52>, ZPZV<4>,
                      ZPZV<3»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type =
04531
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<88>,
                      ZPZV<26>, ZPZV<86»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<89, 12> { using ZPZ = aerobus::zpz<89>; using type =
                      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>
04533
                                     template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04534
                                    template<> struct ConwayPolynomial<89, 17> { using ZPZ = aerobus::zpz<89>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<86»; };</pre>
                     template<> struct ConwayPolynomial<89, 19> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04535
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<86»; }; //</pre>
                      NOLINT
04536
                                     template<> struct ConwayPolynomial<97, 1> { using ZPZ = aerobus::zpz<97>; using type =
                      POLYV<ZPZV<1>, ZPZV<92»; }; // NOLINT
                                 template<> struct ConwayPolynomial<97, 2> { using ZPZ = aerobus::zpz<97>; using type =
04537
                     POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5»; }; // NOLINT
04538
                                     template<> struct ConwayPolynomial<97, 3> { using ZPZ = aerobus::zpz<97>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<92»; }; // NOLINT
                                    template<> struct ConwayPolynomial<97, 4> { using ZPZ = aerobus::zpz<97>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; }; // NOLINT
                                   template<> struct ConwayPolynomial<97, 5> { using ZPZ = aerobus::zpz<97>; using type =
04540
                    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 =
04541
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<58>, ZPZV<88>, ZPZV<58*, }; // NOLINT template<> struct ConwayPolynomial<97, 7> { using ZPZ = aerobus::zpz<97>; using type
04542
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<92»; }; //
04543
                                     template<> struct ConwayPolynomial<97, 8> { using ZPZ = aerobus::zpz<97>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<1>, ZPZV<32>, ZPZV<5»; };
04544
                                   template<> struct ConwayPolynomial<97, 9> { using ZPZ = aerobus::zpz<97>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<59>, ZPZV<7>, ZPZV<92»; };
                      // NOLINT
04545
                                    template<> struct ConwayPolynomial<97, 10> { using ZPZ = aerobus::zpz<97>; using type =
                      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<26>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<20>,
                      ZPZV<5»; }; // NOLINT</pre>
04546
                                    template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<97, 12> { using ZPZ = aerobus::zpz<97>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<59>, ZPZV<81>, ZPZV<81>, ZPZV<86>,
                      ZPZV<78>, ZPZV<94>, ZPZV<5»; }; // NOLINT</pre>
04548
                                     template<> struct ConwayPolynomial<97, 13> { using ZPZ = aerobus::zpz<97>; using type
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    template<> struct ConwayPolynomial<97, 17> { using ZPZ = aerobus::zpz<97>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<92»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<97, 19> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04550
                      NOLINT
04551
                                     template<> struct ConwayPolynomial<101, 1> { using ZPZ = aerobus::zpz<101>; using type =
                      POLYV<ZPZV<1>, ZPZV<99»; }; // NOLINT
04552
                                  template<> struct ConwayPolynomial<101, 2> { using ZPZ = aerobus::zpz<101>; using type =
                     POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
04553
                     POLYY<ZPZY<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
04555
                                     template<> struct ConwayPolynomial<101, 5> { using ZPZ = aerobus::zpz<101>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<99»; }; // NOLINT
                                    template<> struct ConwayPolynomial<101, 6> { using ZPZ = aerobus::zpz<101>; using type =
04556
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<90>, ZPZV<20>, ZPZV<67>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<101,
04557
                                                                                                                                                                                    7> { using ZPZ = aerobus::zpz<101>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<9»; }; // NOLINT
04558
                                  template<> struct ConwayPolynomial<101, 8> { using ZPZ = aerobus::zpz<101>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76>, ZPZV<29>, ZPZV<24>, ZPZV<24>, ZPZV<29; }; //
                     NOLINT
04559
                                   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<0>, ZPZV<64>, ZPZV<47>, ZPZV<499»; };
04560
                                    template<> struct ConwayPolynomial<101, 10> { using ZPZ = aerobus::zpz<101>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<67>, ZPZV<49>, ZPZV<100>, ZPZV<100>, ZPZV<52>,
                       ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<101, 11> { using ZPZ = aerobus::zpz<101>; using type =
04561
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<31>, ZPZV<99»; };</pre>
                                                                                                                   // NOLINT
                                     template<> struct ConwayPolynomial<101, 12> { using ZPZ = aerobus::zpz<101>; using type =
04562
                       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 =
04563
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<99»; };</pre>
                                    template<> struct ConwayPolynomial<101, 17> { using ZPZ = aerobus::zpz<101>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04565
                        ZPZV<0>, ZPZV<0</pre>
04566
                                       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 =
04567
                       POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; }; // NOLINT
                                       template<> struct ConwayPolynomial</103, 3> { using ZPZ = aerobus::zpz<103>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<98»; }; // NOLINT
04569
                                      template<> struct ConwayPolynomial<103, 4> { using ZPZ = aerobus::zpz<103>; using type =
                       template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
04570
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<98»; }; // NOLINT
04571
                                       template<> struct ConwayPolynomial<103, 6> { using ZPZ = aerobus::zpz<103>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<99>, ZPZV<30>, ZPZV<5»; }; // NOLINT
04572
                                    template<> struct ConwayPolynomial<103, 7> { using ZPZ = aerobus::zpz<103>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<98*; }; // NOLINT template<> struct ConwayPolynomial<103, 8> { using ZPZ = aerobus::zpz<103>; using type =
04573
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<70>, ZPZV<71>, ZPZV<49>, ZPZV<5»; }; //
                                       template<> struct ConwayPolynomial<103, 9> { using ZPZ = aerobus::zpz<103>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<98»; };
                        // NOLINT
                      template<> struct ConwayPolynomial<103, 10> { using ZPZ = aerobus::zpz<103>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<101>, ZPZV<86>, ZPZV<101>, ZPZV<94>, ZPZV<11>,
04575
                       ZPZV<5»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<103, 11> { using ZPZ = aerobus::zpz<103>; using type
                       POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                       ZPZV<5>, ZPZV<98»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<103, 12> { using ZPZ = aerobus::zpz<103>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<74>, ZPZV<23>, ZPZV<94>, ZPZV<20>, ZPZV<81>, ZPZV<29>, ZPZV<88>, ZPZV<5»; }; // NOLINT
                                       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»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<103, 17> { using ZPZ = aerobus::zpz<103>; using type =
04579
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 8>, \ \texttt{ZPZV} < 98 \\ \texttt{*}; \ \ // \ \ \texttt{NOLINT} 
                                       template<> struct ConwayPolynomial<103, 19> { using ZPZ = aerobus::zpz<103>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<9»; };</pre>
                       NOLINT
04581
                                     template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
                       POLYV<ZPZV<1>, ZPZV<105»; }; // NOLINT
04582
                                       template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
                       POLYV<ZPZV<1>, ZPZV<103>, ZPZV<2»; }; // NOLINT
04583
                                     template<> struct ConwayPolynomial<107, 3> { using ZPZ = aerobus::zpz<107>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<105»; }; // NOLINT
template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<107, 5> { using ZPZ = aerobus::zpz<107>; using type =
04584
04585
                       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 =
04586
                       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 =
04587
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<105»; }; // NOLINT template<> struct ConwayPolynomial<107, 8> { using ZPZ = aerobus::zpz<107>; using type =
04588
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<24>, ZPZV<95>, ZPZV<2»; };
                       NOLINT
04589
                                     template<> struct ConwayPolynomial<107, 9> { using ZPZ = aerobus::zpz<107>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<66>, ZPZV<105»; };
                        // NOLINT
                                      template<> struct ConwayPolynomial<107, 10> { using ZPZ = aerobus::zpz<107>; using type :
04590
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<94>, ZPZV<61>, ZPZV<83>, ZPZV<83>, ZPZV<85>,
                       ZPZV<2»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<107, 11> { using ZPZ = aerobus::zpz<107>; using type :
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04592
                                    template<> struct ConwayPolynomial<107, 12> { using ZPZ = aerobus::zpz<107>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<48>, ZPZV<6>, ZPZV<6>, ZPZV<61>,
                      ZPZV<42>, ZPZV<57>, ZPZV<2»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105»; };</pre>
                                     /<0>, ZPZV<0>, ZPZV<4>, ZPZV<105»; }; // NOLINT
template<> struct ConwayPolynomial<107, 17> { using ZPZ = aerobus::zpz<107>; using type =
04594
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105»; };</pre>
                                                                                                                                                                                                                                                                                                                           // NOLINT
                                   template<> struct ConwayPolynomial<107, 19> { using ZPZ = aerobus::zpz<107>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      NOLINT
04596
                                     template<> struct ConwayPolynomial<109, 1> { using ZPZ = aerobus::zpz<109>; using type =
                      POLYV<ZPZV<1>, ZPZV<103»; }; // NOLINT
04597
                                   template<> struct ConwayPolynomial<109, 2> { using ZPZ = aerobus::zpz<109>; using type =
                      POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
04598
                     POLYVCZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; }; // NOLINT
                                      template<> struct ConwayPolynomial<109, 5> { using ZPZ = aerobus::zpz<109>; using type =
                      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 =
04601
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<107, ZPZV<102>, ZPZV<66>, ZPZV<68»; }; // NOLINT template<> struct ConwayPolynomial<109, 7> { using ZPZ = aerobus::zpz<109>; using type
04602
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<103»; }; // NOLINT
                                    template<> struct ConwayPolynomial<109, 8> { using ZPZ = aerobus::zpz<109>; using type
04603
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<34>, ZPZV<86>, ZPZV<6»; };
                      NOLINT
04604
                                    template<> struct ConwayPolynomial<109, 9> { using ZPZ = aerobus::zpz<109>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<93>, ZPZV<93>, ZPZV<87>, ZPZV<103»; };
                       // NOLINT
04605
                                     template<> struct ConwayPolynomial<109, 10> { using ZPZ = aerobus::zpz<109>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<71>, ZPZV<55>, ZPZV<16>, ZPZV<75>, ZPZV<69>,
                      ZPZV<6»; }; // NOLINT</pre>
04606
                                     template<> struct ConwayPolynomial<109, 11> { using ZPZ = aerobus::zpz<109>; using type :
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT
                                      template<> struct ConwayPolynomial<109, 12> { using ZPZ = aerobus::zpz<109>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<50>, ZPZV<55>, ZPZV<
                      ZPZV<103>, ZPZV<28>, ZPZV<6»; }; // NOLINT</pre>
04608
                                    template<> struct ConwayPolynomial<109, 13> { using ZPZ = aerobus::zpz<109>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<103»; };</pre>
                                                                                                                                                                                    // NOLINT
                                     template<> struct ConwayPolynomial<109, 17> { using ZPZ = aerobus::zpz<109>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<103»; };</pre>
                                                                                                                                                                                                                                                                                                                        // NOLINT
                                    template<> struct ConwayPolynomial<109, 19> { using ZPZ = aerobus::zpz<109>; using type =
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                      NOLINT
04611
                                      template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
                      POLYV<ZPZV<1>, ZPZV<110»; }; // NOLINT template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
04612
                      POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3»; }; // NOLINT
                                      template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
04613
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<110»; }; // NOLINT
                                    template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
04614
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<62>, ZPZV<3»; }; // NOLINT
04615
                                      template<> struct ConwayPolynomial<113, 5> { using ZPZ = aerobus::zpz<113>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110»; }; // NOLINT template<> struct ConwayPolynomial<113, 6> { using ZPZ = aerobus::zpz<113>; using type =
04616
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<59>, ZPZV<30>, ZPZV<71>, ZPZV<3»; }; // NOLINT
                                       template<> struct ConwayPolynomial<113, 7> { using ZPZ = aerobus::zpz<113>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<110»; };
                      template<> struct ConwayPolynomial<113, 8> { using ZPZ = aerobus::zpz<113>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<38>, ZPZV<28>, ZPZV<28>, ZPZV<3»; }; //</pre>
04618
                      NOLINT
                                    template<> struct ConwayPolynomial<113, 9> { using ZPZ = aerobus::zpz<113>; using type =
04619
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<8110»; };
                                   template<> struct ConwayPolynomial<113, 10> { using ZPZ = aerobus::zpz<113>; using type =
04620
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<55>, ZPZV<45>, ZPZV<83>, ZPZV<56>,
                      ZPZV<3»; }; // NOLINT
                                     template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type =
04621
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                // NOLINT
                      ZPZV<3>, ZPZV<110»; };</pre>
04622
                                    template<> struct ConwayPolynomial<113, 12> { using ZPZ = aerobus::zpz<113>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<23>, ZPZV<62>, ZPZV<4>, ZPZV<98>, ZPZV<56>,
                      ZPZV<10>, ZPZV<27>, ZPZV<3»: }: // NOLINT
                                      template<> struct ConwayPolynomial<113, 13> { using ZPZ = aerobus::zpz<113>; using type :
04623
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                      ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<113, 17> { using ZPZ = aerobus::zpz<113>; using type =
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<110»; }; // NOLINT template<> struct ConwayPolynomial<113, 19> { using ZPZ = aerobus::zpz<113>; using type =
04625
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<2>, ZPZV<22</pre>
                             NOLINT
04626
                                              template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
                             POLYV<ZPZV<1>, ZPZV<124»; }; // NOLINT
                                                template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
04627
                             POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
 04628
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124»; }; // NOLINT
template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<97>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<127, 5> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<3>; }; // NOLINT
Template
 04629
04630
                            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
 04632
                                               template<> struct ConwayPolynomial<127, 7> { using ZPZ = aerobus::zpz<127>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<15>, ZPZV<124»; }; // NOLINT
                                              template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type =
04633
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<104>, ZPZV<55>, ZPZV<8>, ZPZV<3»; }; //
                             template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<119>, ZPZV<126>, ZPZV<124»;
                             }; // NOLINT
                             template<> struct ConwayPolynomial<127, 10> { using ZPZ = aerobus::zpz<127>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<64>, ZPZV<65>, ZPZV<60>, ZPZV<4>,
04635
                             ZPZV<3»; }; // NOLINT
                                             template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type =
04636
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<11>, ZPZV<124»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<127, 12> { using ZPZ = aerobus::zpz<127>; using type =
04637
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<33>, ZPZV<97>, ZPZV<15>, ZPZV<99>, ZPZV<8>, ZPZV<8-, Z
                                                template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type 
04638
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<124»; }; // NOLINT
    template<> struct ConwayPolynomial<127, 17> { using ZPZ = aerobus::zpz<127>; using type =
04639
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             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
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30</pre>
                             NOLINT
                                               template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
04641
                             POLYV<ZPZV<1>, ZPZV<129»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
                             POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2»; }; // NOLINT
 04643
                                              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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<131, 5> { using ZPZ = aerobus::zpz<131>; using type =
04644
04645
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<129»; }; // NOLINT
 04646
                                                template<> struct ConwayPolynomial<131, 6> { using ZPZ = aerobus::zpz<131>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<42, ZPZV<22»; }; // NOLINT template<> struct ConwayPolynomial<131, 7> { using ZPZ = aerobus::zpz<131>; using type =
04647
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<129»; }; // NOLINT template<> struct ConwayPolynomial<131, 8> { using ZPZ = aerobus::zpz<131>; using type =
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<72>, ZPZV<116>, ZPZV<104>, ZPZV<2»; };
04649
                                                template<> struct ConwayPolynomial<131, 9> { using ZPZ = aerobus::zpz<131>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<1>);
                             // NOLINT
04650
                                                template<> struct ConwayPolynomial<131, 10> { using ZPZ = aerobus::zpz<131>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<124>, ZPZV<97>, ZPZV<9>, ZPZV<126>, ZPZV<44>,
                             ZPZV<2»; }; // NOLINT</pre>
04651
                                             template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<6>, ZPZV<129»: }: // NOLINT</pre>
                                              template<> struct ConwayPolynomial<131, 12> { using ZPZ = aerobus::zpz<131>; using type =
04652
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<122>, ZPZV<40>, ZPZV<83>, ZPZV<125>,
                              ZPZV<28>, ZPZV<103>, ZPZV<2»; }; // NOLINT</pre>
04653
                                             template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<129»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
04654
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<62, ZPZV<6</pre>; };
04655
                                              template<> struct ConwayPolynomial<131, 19> { using ZPZ = aerobus::zpz<131>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             NOLINT
                                                template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
                             POLYV<ZPZV<1>, ZPZV<134»; }; // NOLINT
                                              template<> struct ConwayPolynomial<137, 2> { using ZPZ = aerobus::zpz<137>; using type =
                           POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134»; }; // NOLINT
 04658
```

```
template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
04659
                         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<3>; // NOLINT
template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 6> { using ZPZ = aerobus::zpz<137>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<116>, ZPZV<102>, ZPZV<3>, ZPZV<3>; }; // NOLINT template<> struct ConwayPolynomial<137, 7> { using ZPZ = aerobus::zpz<137>; using type =
 04661
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<134»; };
                                       template<> struct ConwayPolynomial<137, 8> { using ZPZ = aerobus::zpz<137>; using type =
                         POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<4>, 2PZV<105>, 2PZV<21>, 2PZV<34>, 2PZV<33); };
                         NOLINT
                                        template<> struct ConwayPolynomial<137, 9> { using ZPZ = aerobus::zpz<137>; using type =
04664
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<80>, ZPZV<122>, ZPZV<134»;
                         }; // NOLINT
                                       template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<20>, ZPZV<20>, ZPZV<67>, ZPZV<93>, ZPZV<119>, ZPZV<3»; }; // NOLINT
04666
                                        template<> struct ConwayPolynomial<137, 11> { using ZPZ = aerobus::zpz<137>; using type :
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<1>, ZPZV<134»; };</pre>
                                                                                                                            // NOLINT
                         template<> struct ConwayPolynomial<137, 12> { using ZPZ = aerobus::zpz<137>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<36>,
                         ZPZV<135>, ZPZV<61>, ZPZV<3»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<137, 13> { using ZPZ = aerobus::zpz<137>; using type =
04668
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       template<> struct ConwayPolynomial<137,</pre>
                                                                                                                                                                                                            17> { using ZPZ = aerobus::zpz<137>; using type =
04669
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<4>, ZPZV<134»; }; // NOLINT
template<> struct ConwayPolynomial<137, 19> { using ZPZ = aerobus::zpz<137>; using type =
04670
                         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<134w; }; //</pre>
                                        template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
04671
                         POLYV<ZPZV<1>, ZPZV<137»; }; // NOLINT
                                          template<> struct ConwayPolynomial<139, 2> { using ZPZ = aerobus::zpz<139>; using type =
04672
                         POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<139, 3> { using ZPZ = aerobus::zpz<139>; using type =
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<137»; }; // NOLINT template<> struct ConwayPolynomial<139, 4> { using ZPZ = aerobus::zpz<139>; using type =
 04674
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<139, 5> { using ZPZ = aerobus::zpz<139>; using type =
04675
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<137»; }; // NOLINT
 04676
                                         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
 04677
                                          template<> struct ConwayPolynomial<139, 7> { using ZPZ = aerobus::zpz<139>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<3>, ZPZV<137»; }; // NOLINT template<> struct ConwayPolynomial<139, 8> { using ZPZ = aerobus::zpz<139>; using type =
 04678
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>, ZPZV<36>, ZPZV<21>, ZPZV<2»; };
                         NOLINT
04679
                                          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<3>, ZPZV<70>, ZPZV<87>, ZPZV<137»; };
                          // NOLINT
04680
                                          template<> struct ConwayPolynomial<139, 10> { using ZPZ = aerobus::zpz<139>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPŽV<0>, ZPZV<0>, ZPZV<0>, ZPZV<110>, ZPZV<48>, ZPZV<130>, ZPŽV<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<0
                          ZPZV<7>, ZPZV<137»; }; // NOLINT</pre>
04682
                                          template<> struct ConwayPolynomial<139, 12> { using ZPZ = aerobus::zpz<139>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<100>, ZPZV<120>, ZPZV<15>, ZPZV<41>, ZPZV<77>, ZPZV<106>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
04683
                                          template<> struct ConwayPolynomial<139, 13> { using ZPZ = aerobus::zpz<139>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                    // NOLINT
                          ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<137»; };</pre>
04684
                                       template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type =
                          \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<137»; ); // NOLINT template<> struct ConwayPolynomial<139, 19> { using ZPZ = aerobus::zpz<139>; using type
04685
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<137»; }; //</pre>
04686
                                          template<> struct ConwayPolynomial<149, 1> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<147»; }; // NOLINT
04687
                                          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 =
 04688
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
04689
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<149, 5> { using ZPZ = aerobus::zpz<149>; using type =
 04690
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<147»; }; // NOLINT
                                          template<> struct ConwayPolynomial<149, 6> { using ZPZ = aerobus::zpz<149>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<105>, ZPZV<33>, ZPZV<55>, ZPZV<2»; }; // NOLINT
 04692
                                       template<> struct ConwayPolynomial<149, 7> { using ZPZ = aerobus::zpz<149>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV
 04693
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<140>, ZPZV<25>, ZPZV<123>, ZPZV<2»; }; //
                                              template<> struct ConwayPolynomial<149, 9> { using ZPZ = aerobus::zpz<149>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<146>, ZPZV<146>, ZPZV<20>, ZPZV<147»;
                              }; // NOLINT
                                                  template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type =
04695
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<143>, ZPZV<143>, ZPZV<143>, ZPZV<151>,
                              ZPZV<2»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<149, 11> { using ZPZ = aerobus::zpz<149>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type
04697
                             POLYV<ZPZV<1>, ZPZV<0>, 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
                                              template<> struct ConwayPolynomial<149, 13> { using ZPZ = aerobus::zpz<149>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type :
04699
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<147»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<149, 19> { using ZPZ = aerobus::zpz<149>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               \texttt{ZPZV} < \texttt{0} >, \ \texttt{Z
                              NOLINT
04701
                                                 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 =
04702
                             POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6»; }; // NOLINT
04703
                                                 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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6»; }; // NOLINT
04704
                                                template<> struct ConwayPolynomial<151, 5> { using ZPZ = aerobus::zpz<151>; using type =
04705
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<145»; }; // NOLINT
04706
                                                template<> struct ConwayPolynomial<151, 6> { using ZPZ = aerobus::zpz<151>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<125>, ZPZV<15>, ZPZV<15>, ZPZV<6>; // NOLINT template<> struct ConwayPolynomial<151, 7> { using ZPZ = aerobus::zpz<151>; using type =
04707
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<145»; }; // NOLINT
                                                template<> struct ConwayPolynomial<151, 8> { using ZPZ = aerobus::zpz<151>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<140>, ZPZV<122>, ZPZV<43>, ZPZV<6»; }; //
                            template<> struct ConwayPolynomial<151, 9> { using ZPZ = aerobus::zpz<151>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<126>, ZPZV<126>, ZPZV<126>, ZPZV<145»;
04709
                            }; // NOLINT
  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<0
                              ZPZV<1>, ZPZV<145»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<151, 12> { using ZPZ = aerobus::zpz<151>; using type =
                              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<109>, ZPZV<101>, ZPZV<101>, ZPZV<60>, ZPZV<7>,
                              ZPZV<107>, ZPZV<147>, ZPZV<6»; }; // NOLINT</pre>
04713
                                                 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>
                                                  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>
04715
                                                 \texttt{template<> struct ConwayPolynomial<151, 19> \{ using \ ZPZ = aerobus:: zpz<151>; \ using \ type = aerobus:: zpz<151>; \
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              NOLINT
                                                 template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
                             POLYV<ZPZV<1>, ZPZV<152»; }; // NOLINT
04717
                                               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 =
04718
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<152»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
04719
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<136>, ZPZV<5»; }; // NOLINT
04720
                                              template<> struct ConwayPolynomial<157, 5> { using ZPZ = aerobus::zpz<157>; using type =
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, SPZV<7>, SPZV<7
, SPZV
04721
                            POLYV<ZPZV<1>, ZPZV<3>, ZPZV<130>, ZPZV<43>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<157, 7> { using ZPZ = aerobus::zpz<157>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; };
04723
                                              template<> struct ConwayPolynomial<157, 8> { using ZPZ = aerobus::zpz<157>; using type =
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<97>, ZPZV<40>, ZPZV<153>, ZPZV<5»; }; //
                             NOLINT
04724
                                                template<> struct ConwayPolynomial<157, 9> { using ZPZ = aerobus::zpz<157>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<114>, ZPZV<52>, ZPZV<152»;
                              }; // NOLINT
                                               template<> struct ConwayPolynomial<157, 10> { using ZPZ = aerobus::zpz<157>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<61>, ZPZV<22>, ZPZV<124>, ZPZV<61>, ZPZV<93>, ZPZV<5»; }; // NOLINT
                                              template<> struct ConwayPolynomial<157, 11> { using ZPZ = aerobus::zpz<157>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                         ZPZV<29>, ZPZV<152»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<157, 12> { using ZPZ = aerobus::zpz<157>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<77>, ZPZV<110>, ZPZV<72>, ZPZV<137>, ZPZV<43>,
                         ZPZV<152>, ZPZV<57>, ZPZV<5»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<157, 13> { using ZPZ = aerobus::zpz<157>; using type =
04728
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         template<> struct ConwayPolynomial<157, 17> { using ZPZ = aerobus::zpz<157>; using type =
04729
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<152»; }; // NOLINT template<> struct ConwayPolynomial<157, 19> { using ZPZ = aerobus::zpz<157>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                                 ZPZV<0>.
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; }; //</pre>
                         NOLINT
04731
                                         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 =
04732
                         POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
                        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<7>, ZPZY<161»; }; // NOLINT template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
04734
                        template<> struct ConwayPolynomial<163, 5> { using ZPZ = aerobus::zpz<163>; using type =
04735
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<161»; }; // NOLINT
                                          template<> struct ConwayPolynomial<163, 6> { using ZPZ = aerobus::zpz<163>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<25>, ZPZV<156>, ZPZV<2»; }; // NOLINT
04737
                                        template<> struct ConwayPolynomial<163, 7> { using ZPZ = aerobus::zpz<163>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, Z
04738
                                        template<> struct ConwayPolynomial<163, 8> { using ZPZ = aerobus::zpz<163>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<132>, ZPZV<83>, ZPZV<6>, ZPZV<2»; }; //
                        NOLINT
                                         template<> struct ConwayPolynomial<163, 9> { using ZPZ = aerobus::zpz<163>; using type =
04739
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<162>, ZPZV<161»;
                         }; // NOLINT
04740
                                         template<> struct ConwayPolynomial<163, 10> { using ZPZ = aerobus::zpz<163>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<111>, ZPZV<120>, ZPZV<125>, ZPZV<15>, ZPZV<0>,
                         ZPZV<2»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<163, 11> { using ZPZ = aerobus::zpz<163>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<11>, ZPZV<161»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type =
04742
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<112>, ZPZV<31>, ZPZV<38>, ZPZV<103>,
                         ZPZV<10>, ZPZV<69>, ZPZV<2»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<163, 13> { using ZPZ = aerobus::zpz<163>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<161»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<163, 17> { using ZPZ = aerobus::zpz<163>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<161»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<163, 19> { using ZPZ = aerobus::zpz<163>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0</pre>; };
                         NOLINT
                                         template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
                        POLYV<ZPZV<1>, ZPZV<162»; }; // NOLINT
                                          template<> struct ConwayPolynomial<167, 2> { using ZPZ = aerobus::zpz<167>; using type =
                       POLYV<ZPZV<1>, ZPZV<166>, ZPZV<5»; }; // NOLINT
04748
                                          template<> struct ConwayPolynomial<167, 3> { using ZPZ = aerobus::zpz<167>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162»; ); // NOLINT
template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 5> { using ZPZ = aerobus::zpz<167>; using type =
04749
04750
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<162»; }; // NOLINT
04751
                                        template<> struct ConwayPolynomial<167, 6> { using ZPZ = aerobus::zpz<167>; using type =
                         \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 2>, \ \texttt{ZPZV} < 75>, \ \texttt{ZPZV} < 38>, \ \texttt{ZPZV} < 2>, \ \texttt{ZPZV} < 5>; \ \}; \ \ // \ \texttt{NOLINT} 
                                         template<> struct ConwayPolynomial<167, 7> { using ZPZ = aerobus::zpz<167>; using type
04752
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<162»; }; // NOLINT
                                       template<> struct ConwayPolynomial<167, 8> { using ZPZ = aerobus::zpz<167>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<149>, ZPZV<56>, ZPZV<113>, ZPZV<5»; };
04754
                                      template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<165>, ZPZV<122>, ZPZV<162»;
                         }; // NOLINT
04755
                                          template<> struct ConwayPolynomial<167, 10> { using ZPZ = aerobus::zpz<167>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68>, ZPZV<68>, ZPZV<109>, ZPZV<143>,
                                                                                                                            // NOLINT
                         ZPZV<148>, ZPZV<5»; };</pre>
04756
                                        template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<24>, ZPZV<162»; }; // NOLINT template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type =
04757
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<142>, ZPZV<142>, ZPZV<142>, ZPZV<131>,
                        ZPZV<140>, ZPZV<41>, ZPZV<57>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 13> { using ZPZ = aerobus::zpz<167>; using type :
                         \texttt{POLYV} < \texttt{ZPZV} < \texttt{0} >, \ \texttt{ZPZV} < \texttt{
                        ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162»; }; // NOLINT
template<> struct ConwayPolynomial<167, 17> { using ZPZ = aerobus::zpz<167>; using type =
04759
```

```
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<6>, ZPZV<6>, ZPZV<62>, ZPZV<62»; }; // NOLINT
template<> struct ConwayPolynomial<167, 19> { 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<14>, ZPZV<162»; }; //</pre>
                         NOLINT
                                          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<2w; }; // NOLINT template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
04763
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
04764
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<2»; }; // NOLINT
04765
                                        template<> struct ConwayPolynomial<173, 5> { using ZPZ = aerobus::zpz<173>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 6> { using ZPZ = aerobus::zpz<173>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<134>, ZPZV<107>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<173, 7> { using ZPZ = aerobus::zpz<173>; using type =
04766
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<171»; };
                                         template<> struct ConwayPolynomial<173, 8> { using ZPZ = aerobus::zpz<173>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<125>, ZPZV<158>, ZPZV<27>, ZPZV<2»; }; //
                                        template<> struct ConwayPolynomial<173, 9> { using ZPZ = aerobus::zpz<173>; using type =
04769
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<104>, ZPZV<171»;
                         }; // NOLINT template<> struct ConwayPolynomial<173, 10> { using ZPZ = aerobus::zpz<173>; using type =
04770
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<156>, ZPZV<164>, ZPZV<48>, ZPZV<106>,
                         ZPZV<58>, ZPZV<2»; }; // NOLINT
    template<> struct ConwayPolynomial<173, 11> { using ZPZ = aerobus::zpz<173>; using type =
04771
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<12>, ZPZV<171»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type
04772
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<64>, ZPZV<46>, ZPZV<166>, ZPZV<0>,
                         \mbox{ZPZV}<159> , \mbox{ZPZV}<22> , \mbox{ZPZV}<2 ); // NOLINT
                                           template<> struct ConwayPolynomial<173, 13> { using ZPZ = aerobus::zpz<173>; using type =
04773
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                 // NOLINT
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<171»; };</pre>
                         template<> struct ConwayPolynomial<173, 19> { using ZPZ = aerobus::zpz<173>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0>, ZPZV<0>, ZPZV<0 , ZPZV<0
04775
                          ZPZV<0>, ZPZV<6</pre>, ZPZV<6</pre>
                         NOLINT
04776
                                          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
                                          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 =
04779
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<2»; }; // NOLINT
04780
                                         template<> struct ConwayPolynomial<179, 5> { using ZPZ = aerobus::zpz<179>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<177»; }; // NOLINT
                                          template<> struct ConwayPolynomial<179, 6> { using ZPZ = aerobus::zpz<179>; using type =
04781
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<91>, ZPZV<55>, ZPZV<109>, ZPZV<2»; }; // NOLINT
                                       template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
04782
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>; // NOLINT template<> struct ConwayPolynomial<179, 8> { using ZPZ = aerobus::zpz<179>; using type =
04783
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<163>, ZPZV<144>, ZPZV<73>, ZPZV<2»; }; //
                         NOLINT
                                          template<> struct ConwayPolynomial<179, 9> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<54>, ZPZV<54
                         template<> struct ConwayPolynomial<179, 10> { using ZPZ = aerobus::zpz<179>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<115>, ZPZV<71>, ZPZV<150>, ZPZV<49>, ZPZV<87>,
04785
                         ZPZV<2»: }: // NOLINT</pre>
                                         template<> struct ConwayPolynomial<179, 11> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<28>, ZPZV<177»; }; // NOLINT</pre>
04787
                                       template<> struct ConwayPolynomial<179, 12> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<103>, ZPZV<83>, ZPZV<43>, ZPZV<76>, ZPZV<8>, ZPZV<177>, ZPZV<1>, ZPZV<1>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<179, 13> { using ZPZ = aerobus::zpz<179>; using type =
04788
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<177»; }; // NOLINT</pre>
04789
                                        template<> struct ConwayPolynomial<179,
                                                                                                                                                                                                                   17> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177»; ); // NOLINT
template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type
04790
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<11>, ZPZV<17**, }; //</pre>
                         NOLINT
04791
                                         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»; };
                                    template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
 04793
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179»; }; // NOLINT template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
 04794
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<105>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
04795
                    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 =
 04796
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<163>, ZPZV<169>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type =
 04797
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>; // NOLINT template<> struct ConwayPolynomial<181, 8> { using ZPZ = aerobus::zpz<181>; using type =
04798
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<108>, ZPZV<22>, ZPZV<149>, ZPZV<2»; }; //
 04799
                                 template<> struct ConwayPolynomial<181, 9> { using ZPZ = aerobus::zpz<181>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<107>, ZPZV<168>, ZPZV<179»;
                      }; // NOLINT
04800
                                    template<> struct ConwayPolynomial<181, 10> { using ZPZ = aerobus::zpz<181>; using type :
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<154>, ZPZV<104>, ZPZV<94>, ZPZV<57>, ZPZV<88>,
                      ZPZV<2»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<181, 11> { using ZPZ = aerobus::zpz<181>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<24>, ZPZV<179»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<181, 12> { using ZPZ = aerobus::zpz<181>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<141>, ZPZV<45>, ZPZV<12>, ZPZV<175>, ZPZV<12>, ZPZV<10>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<181, 13> { using ZPZ = aerobus::zpz<181>; using type =
04802
04803
                      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 =
04804
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<179»; };</pre>
                                                                                                                                                                                                                                                                                                         // NOLINT
                                    template<> struct ConwayPolynomial<181, 19> { using ZPZ = aerobus::zpz<181>; using type =
04805
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<179»; }; //</pre>
                      NOLINT
04806
                                   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 =
                      POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19»; }; // NOLINT
04808
                                  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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; }; // NOLINT
 04809
                                    template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<172»; }; // NOLINT
 04811
                                  template<> struct ConwayPolynomial<191, 6> { using ZPZ = aerobus::zpz<191>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<19»; }; // NOLINT template<> struct ConwayPolynomial<191, 7> { using ZPZ = aerobus::zpz<191>; using type =
04812
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<172»; }; // NOLINT
                                    template<> struct ConwayPolynomial<191, 8> { using ZPZ = aerobus::zpz<191>; using type
04813
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<164>, ZPZV<139>, ZPZV<171>, ZPZV<19»; }; //
                      NOLINT
04814
                                  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<1>3>, ZPZV<113>, ZPZV<47>, ZPZV<173>, ZPZV<74>,
                      ZPZV<156>, ZPZV<19»; }; // NOLINT</pre>
04816
                                    template<> struct ConwayPolynomial<191, 11> { using ZPZ = aerobus::zpz<191>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    template<> struct ConwayPolynomial<191, 12> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<168>, ZPZV<25>, ZPZV<49>, ZPZV<90>,
                      ZPZV<7>, ZPZV<151>, ZPZV<19»; }; // NOLINT</pre>
04818
                                 template<> struct ConwayPolynomial<191, 13> { using ZPZ = aerobus::zpz<191>; using type =
                      \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                     ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<172»; }; // NOLINT template<> struct ConwayPolynomial<191, 17> { using ZPZ = aerobus::zpz<191>; using type :
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<172»; };</pre>
04820
                                 template<> struct ConwayPolynomial<191, 19> { using ZPZ = aerobus::zpz<191>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<190>, ZPZV<2>, ZPZV<172»; }; //
                      NOLINT
                                    template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
                     POLYV<ZPZV<1>, ZPZV<188»; }; // NOLINT
                                  template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
                    POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5»; }; // NOLINT
                                   template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
04823
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
 04824
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<5»; }; // NOLINT
 04825
                                  template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
                    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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<8>, ZPZV<172>, ZPZV<5»; }; // NOLINT
 04826
```

```
04827
                                          template<> struct ConwayPolynomial<193, 7> { 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<3>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 8> { using ZPZ = aerobus::zpz<193>; using type =
04828
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<145>, ZPZV<34>, ZPZV<154>, ZPZV<5»; }; //
                        NOLINT
                                        template<> struct ConwayPolynomial<193, 9> { using ZPZ = aerobus::zpz<193>; using type =
04829
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<168>, ZPZV<17>, ZPZV<188»;
                        }; // NOLINT
                                       template<> struct ConwayPolynomial<193, 10> { using ZPZ = aerobus::zpz<193>; using type =
04830
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<51>, ZPZV<77>, ZPZV<0>, ZPZV<89>,
                        ZPZV<5»; }; // NOLINT
                                       template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type :
04831
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      template<> struct ConwayPolynomial<193, 12> { using ZPZ = aerobus::zpz<193>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<5>, ZPZV<52>, ZPZV<135>, ZPZV<152>,
                        ZPZV<90>, ZPZV<46>, ZPZV<28>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<193, 13> { usi
04833
                                                                                                                                                                                                           13> { using ZPZ = aerobus::zpz<193>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<188»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<193, 17> { using ZPZ = aerobus::zpz<193>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<188»; }; // NOLINT
    template<> struct ConwayPolynomial<193, 19> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04835
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<188»; }; //</pre>
04836
                                        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 =
04837
                        POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2»; }; // NOLINT
04838
                                         template<> struct ConwayPolynomial<197, 3> { using ZPZ = aerobus::zpz<197>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<195»; }; // NOLINT
                                      template<> struct ConwayPolynomial<197, 4> { using ZPZ = aerobus::zpz<197>; using type =
04839
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<124>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<197, 5> { using ZPZ = aerobus::zpz<197>; using type =
04840
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195»; // NOLINT
                                         template<> struct ConwayPolynomial<197, 6> { using ZPZ = aerobus::zpz<197>; using type =
                        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 =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                        template<> struct ConwayPolynomial<197, 8> { using ZPZ = aerobus::zpz<197>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZV<0 ; ZPZV<0 
04843
                                         template<> struct ConwayPolynomial<197, 9> { using ZPZ = aerobus::zpz<197>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<127>, ZPZV<8>, ZPZV<195»;
                        }; // NOLINT
                                         template<> struct ConwayPolynomial<197, 10> { using ZPZ = aerobus::zpz<197>; using type :
04845
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<137>, ZPZV<8>, ZPZV<73>, ZPZV<42>,
                        ZPZV<2»; }; // NOLINT</pre>
04846
                                         template<> struct ConwayPolynomial<197, 11> { using ZPZ = aerobus::zpz<197>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<14>, ZPZV<195»; }; // NOLINT</pre>
04847
                                         template<> struct ConwayPolynomial<197, 12> { using ZPZ = aerobus::zpz<197>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>,
                        ZPZV<90>, ZPZV<163>, ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<197, 13> { using ZPZ = aerobus::zpz<197>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<195»; }; // NOLINT</pre>
04849
                                         template<> struct ConwayPolynomial<197,
                                                                                                                                                                                                           17> { using ZPZ = aerobus::zpz<197>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04850
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0</pre>
                        NOLINT
04851
                                         template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
                        POLYV<ZPZV<1>, ZPZV<196»; }; // NOLINT
                                         template<> struct ConwayPolynomial<199, 2> { using ZPZ = aerobus::zpz<199>; using type =
04852
                        POLYV<ZPZV<1>, ZPZV<193>, ZPZV<3»; }; // NOLINT
                                         template<> struct ConwayPolynomial<199, 3> { using ZPZ = aerobus::zpz<199>; using type =
04853
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; }; // NOLINT template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
04854
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
04855
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
                                       template<> struct ConwayPolynomial<199, 6> { using ZPZ = aerobus::zpz<199>; using type =
04856
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<90>, ZPZV<58>, ZPZV<79>, ZPZV<3»; }; // NOLINT
04857
                                        template<> struct ConwayPolynomial<199, 7> { using ZPZ = aerobus::zpz<199>; using t
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
                                       template<> struct ConwayPolynomial<199, 8> { using ZPZ = aerobus::zpz<199>; using type =
04858
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<160>, ZPZV<23>, ZPZV<159>, ZPZV<159>, ZPZV<39; }; //
04859
                                       template<> struct ConwayPolynomial<199, 9> { using ZPZ = aerobus::zpz<199>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<177>, ZPZV<141>, ZPZV<196»;
                        }; // NOLINT
04860
                                      template<> struct ConwayPolynomial<199, 10> { using ZPZ = aerobus::zpz<199>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<158>, ZPZV<31>, ZPZV<54>, ZPZV<9>,
                      ZPZV<3»; }; // NOLINT</pre>
                                  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>
                                    template<> struct ConwayPolynomial<199, 12> { using ZPZ = aerobus::zpz<199>; using type =
04862
                      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</pre>
                                  template<> struct ConwayPolynomial<199, 13> { using ZPZ = aerobus::zpz<199>; using type =
04863
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  template<> struct ConwayPolynomial<199, 17> { using ZPZ = aerobus::zpz<199>; using type
04864
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<196»; };</pre>
                                 template<> struct ConwayPolynomial<199, 19> { using ZPZ = aerobus::zpz<199>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                                                                                                                                                                                                                                                                                                                             7.P7.V<0>.
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<196»; }; //
                      NOLINT
                                    template<> struct ConwayPolynomial<211, 1> { using ZPZ = aerobus::zpz<211>; using type =
                     POLYV<ZPZV<1>, ZPZV<209»; }; // NOLINT
                                    template<> struct ConwayPolynomial<211, 2> { using ZPZ = aerobus::zpz<211>; using type =
                     POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; }; // NOLINT
                                  template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
04868
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<209»; }; // NOLINT template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
04869
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2»; }; // NOLINT
04870
                                  template<> struct ConwayPolynomial<211, 5> { using ZPZ = aerobus::zpz<211>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<209»; }; // NOLINT
04871
                                    template<> struct ConwayPolynomial<211, 6> { using ZPZ = aerobus::zpz<211>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<81>, ZPZV<194>, ZPZV<133>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<211, 7> { using ZPZ = aerobus::zpz<211>; using type = aerobus::zpz<2
04872
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<209»; };
                                    template<> struct ConwayPolynomial<211, 8> { using ZPZ = aerobus::zpz<211>; using type
04873
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<200>, ZPZV<87>, ZPZV<29>, ZPZV<2»; };
                      NOLINT
                                   template<> struct ConwayPolynomial<211, 9> { using ZPZ = aerobus::zpz<211>; using type =
04874
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<139>, ZPZV<139>, ZPZV<26>, ZPZV<209»;
                                     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<87>, ZPZV<125>,
                      ZPZV<2»; }; // NOLINT</pre>
04876
                                  template<> struct ConwayPolynomial<211, 11> { using ZPZ = aerobus::zpz<211>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<7>, ZPZV<209»; };</pre>
                                                                                                           // NOLINT
                                    template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<50>, ZPZV<145>, ZPZV<126>, ZPZV<184>,
                      ZPZV<84>, ZPZV<27>, ZPZV<2»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<211, 13> { using ZPZ = aerobus::zpz<211>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<209»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<211,
04879
                                                                                                                                                                                   17> { 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<12>, ZPZV<209»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<211, 19> { using ZPZ = aerobus::zpz<211>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
04880
                      ZPZV<0>, ZPZV<17>, ZPZV<209»; }; //</pre>
                                   template<> struct ConwayPolynomial<223, 1> { using ZPZ = aerobus::zpz<223>; using type =
                      POLYV<ZPZV<1>, ZPZV<220»; }; // NOLINT
04882
                                    template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
                     POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; }; // NOLINT
                                  template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
04883
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<220»; }; // NOLINT
                                    template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<3»; }; // NOLINT
                                  template<> struct ConwayPolynomial<223, 5> { using ZPZ = aerobus::zpz<223>; using type =
04885
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2), ZPZV<2), ZPZV<20, ZPZV<20»; }; // NOLINT template<> struct ConwayPolynomial<223, 6> { using ZPZ = aerobus::zpz<223>; using type =
04886
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68>, ZPZV<24>, ZPZV<196>, ZPZV<3»; }; // NOLINT
                                    template<> struct ConwayPolynomial<223, 7> { using ZPZ = aerobus::zpz<223>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
04888
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<3»; }; //
                      NOLINT
                                  template<> struct ConwayPolynomial<223, 9> { using ZPZ = aerobus::zpz<223>; using type
04889
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<164>, ZPZV<64>, ZPZV<220»;
                      }; // NOLINT
04890
                                    template<> struct ConwayPolynomial<223, 10> { using ZPZ = aerobus::zpz<223>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<118>, ZPZV<177>, ZPZV<87>, ZPZV<99>, ZPZV<62>,
                      ZPZV<3»: }: // NOLINT
                                    template<> struct ConwayPolynomial<223, 11> { using ZPZ = aerobus::zpz<223>; using type :
04891
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                      ZPZV<8>, ZPZV<220»; }; // NOLINT</pre>
04892
                                    template<> struct ConwayPolynomial<223, 12> { using ZPZ = aerobus::zpz<223>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<64>, ZPZV<94>, ZPZV<11>, ZPZV<105>, ZPZV<64>,
                      ZPZV<151>, ZPZV<213>, ZPZV<3»; }; // NOLINT</pre>
04893
                                  template<> struct ConwayPolynomial<223, 13> { using ZPZ = aerobus::zpz<223>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                             ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<220»; };  // NOLINT</pre>
                                            template<> struct ConwayPolynomial<223, 17> { using ZPZ = aerobus::zpz<223>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; }; // NOLINT
template<> struct ConwayPolynomial<223, 19> { using ZPZ = aerobus::zpz<223>; using type =
04895
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>
                                             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 =
04897
                           POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2»; }; // NOLINT
                                                template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, 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
 04899
                                               template<> struct ConwayPolynomial<227, 5> { using ZPZ = aerobus::zpz<227>; using type =
04900
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<225»; }; // NOLINT
                                               template<> struct ConwayPolynomial<227, 6> { using ZPZ = aerobus::zpz<227>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<24>, ZPZV<135>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<227, 7> { using ZPZ = aerobus::zpz<227>; using type =
 04902
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25, ZPZV<2
04903
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<151>, ZPZV<176>, ZPZV<106>, ZPZV<2»; }; //
                                            template<> struct ConwayPolynomial<227, 9> { using ZPZ = aerobus::zpz<227>; using type =
04904
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<183>, ZPZV<225»;
                             }; // NOLINT
04905
                                              template<> struct ConwayPolynomial<227, 10> { using ZPZ = aerobus::zpz<227>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<199>, ZPZV<12>, ZPZV<12>, ZPZV<93>, ZPZV<77>,
                             ZPZV<2»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type
04906
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<2>, ZPZV<225»; }; // NOLINT</pre>
04907
                                               template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; 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<0, ZPZV<0>, ZPZV<123>, ZPZV<96>, 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
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<O>, ZPZV<Z>, ZPZV<Z>, ZPZV<Z>>; ;; // NOLINI
    template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
04909
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>; // NOLINT
template<> struct ConwayPolynomial<227, 19> { using ZPZ = aerobus::zpz<227>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<34>, ZPZV<225»; }; //</pre>
                             NOLINT
04911
                                              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 =
04912
                             POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»; }; // NOLINT
 04913
                                               template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; }; // NOLINT template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
04914
                           POLYYCZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<6w; }; // NOLINT template<> struct ConwayPolynomial<229, 5> { using ZPZ = aerobus::zpz<229>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223»; }; // NOLINT
                                               template<> struct ConwayPolynomial<229, 6> { using ZPZ = aerobus::zpz<229>; using type =
 04916
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<160>, ZPZV<186>, ZPZV<6»; }; // NOLINT
                           template<> struct ConwayPolynomial<229, 7> { using ZPZ = aerobus::zpz<229>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<23»; }; // NOLINT
04917
                                              template<> struct ConwayPolynomial<229, 8> { using ZPZ = aerobus::zpz<229>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<193>, ZPZV<62>, ZPZV<205>, ZPZV<6»; };
04919
                                           template<> struct ConwayPolynomial<229, 9> { using ZPZ = aerobus::zpz<229>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<117>, ZPZV<117>, ZPZV<50>, ZPZV<223»;
                             }; // NOLINT
                                                template<> struct ConwayPolynomial<229, 10> { using ZPZ = aerobus::zpz<229>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<185>, ZPZV<135>, ZPZV<158>, ZPZV<167>,
                             ZPZV<98>, ZPZV<6»; };</pre>
                                                                                                                                      // NOLINT
04921
                                           template<> struct ConwayPolynomial<229, 11> { using ZPZ = aerobus::zpz<229>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<223»; }; // NOLINT
                                               template<> struct ConwayPolynomial<229, 12> { using ZPZ = aerobus::zpz<229>; using type =
04922
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<131>, ZPZV<140>, ZPZV<25>, ZPZV<172>, ZPZV<9>, ZPZV<6>; }; // NOLINT
04923
                                            template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                               template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type =
04924
                             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<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; ; // NOLINT
template<> struct ConwayPolynomial<229, 19> { using ZPZ = aerobus::zpz<229>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<223»; }; //</pre>
                             NOT.TNT
```

```
04926
                                               template<> struct ConwayPolynomial<233, 1> { using ZPZ = aerobus::zpz<233>; using type =
                           POLYV<ZPZV<1>, ZPZV<230»; }; // NOLINT
                                           template<> struct ConwayPolynomial<233, 2> { using ZPZ = aerobus::zpz<233>; using type =
                           POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
 04928
                                             template<> struct ConwayPolynomial<233, 3> { using ZPZ = aerobus::zpz<233>; using type =
                          POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<3»; }; // NOLINT
 04930
                                           template<> struct ConwayPolynomial<233, 5> { using ZPZ = aerobus::zpz<233>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<230»; }; // NOLINT
04931
                                            \texttt{template<> struct ConwayPolynomial<233, 6> \{ using \ ZPZ = aerobus:: zpz<233>; \ using \ type = aerobus:: zpz<233>; \ 
                            \verb|Polyv<zpzv<1>, & zpzv<0>, & zpzv<3>, & zpzv<122>, & zpzv<215>, & zpzv<32>, & zpzv<3»; & || // & Nolint | || N
04932
                                             template<> struct ConwayPolynomial<233,
                                                                                                                                                                                                                              7> { using ZPZ = aerobus::zpz<233>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<230»; }; //
                                          template<> struct ConwayPolynomial<233, 8> { using ZPZ = aerobus::zpz<233>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<202>, ZPZV<135>, ZPZV<181>, ZPZV<3»; }; //
                            NOLTNT
                           template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<56>, ZPZV<146>, ZPZV<230»;</pre>
04934
                           }; // NOLINT
  template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<28>, ZPZV<71>, ZPZV<102>, ZPZV<102, ZPZV<48>,
                            ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<233, 11> { using ZPZ = aerobus::zpz<233>; using type =
04936
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<5>, ZPZV<230»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<233, 12> { using ZPZ = aerobus::zpz<233>; using type
04937
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, 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 =
04938
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<230»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<233, 17> { using ZPZ = aerobus::zpz<233>; using type =
04939
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43, ZPZV<230»; }; // NOLINT
template<> struct ConwayPolynomial<233, 19> { using ZPZ = aerobus::zpz<233>; using type =
04940
                            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<25>, ZPZV<25>, ZPZV<230»; }; //</pre>
                                              template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
                           POLYV<ZPZV<1>, ZPZV<232»; }; // NOLINT
                                             template<> struct ConwayPolynomial<239, 2> { using ZPZ = aerobus::zpz<239>; using type =
04942
                           POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7»: }: // NOLINT
04943
                                             template<> struct ConwayPolynomial<239, 3> { using ZPZ = aerobus::zpz<239>; using type =
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<232»; }; // NOLINT template<> struct ConwayPolynomial<239, 4> { using ZPZ = aerobus::zpz<239>; using type =
 04944
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7»; }; // NOLINT
                                            template<> struct ConwayPolynomial<239, 5> { using ZPZ = aerobus::zpz<239>; using type =
04945
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; }; // NOLINT
                                             template<> struct ConwayPolynomial<239, 6> { using ZPZ = aerobus::zpz<239>; using type =
 04946
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<237>, ZPZV<60>, ZPZV<200>, ZPZV<7»; };
                                              template<> struct ConwayPolynomial<239, 7> { using ZPZ = aerobus::zpz<239>, using type =
 04947
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<232»; };
04948
                                            template<> struct ConwayPolynomial<239, 8> { using ZPZ = aerobus::zpz<239>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<202>, ZPZV<54>, ZPZV<7»; }; //
                           NOLINT
                                             template<> struct ConwayPolynomial<239, 9> { using ZPZ = aerobus::zpz<239>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<88>, ZPZV<232»; };
04950
                                            template<> struct ConwayPolynomial<239, 10> { using ZPZ = aerobus::zpz<239>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<68>, ZPZV<226>, ZPZV<127>, ZPZV<108>, ZPZV<7»; }; // NOLINT
                                             template<> struct ConwayPolynomial<239, 11> { using ZPZ = aerobus::zpz<239>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                     // NOLINT
                            ZPZV<8>, ZPZV<232»; };</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<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 =
04952
04953
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; };</pre>
                                                                                                                                                                                                                     // NOLINT
04954
                                          template<> struct ConwayPolynomial<239, 17> { using ZPZ = aerobus::zpz<239>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04955
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<232»; }; //</pre>
04956
                                             template<> struct ConwayPolynomial<241, 1> { using ZPZ = aerobus::zpz<241>; using type =
                           POLYV<ZPZV<1>. ZPZV<234»: }: // NOLINT
                                             template<> struct ConwayPolynomial<241, 2> { using ZPZ = aerobus::zpz<241>; using type =
 04957
                          POLYV<ZPZV<1>, ZPZV<238>, ZPZV<7»; }; // NOLINT
                                             template<> struct ConwayPolynomial<241, 3> { using ZPZ = aerobus::zpz<241>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<234»; };
                                                                                                                                                                                                                                                 // NOLINT
                                          template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
 04959
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<152, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<241, 5> { using ZPZ = aerobus::zpz<241>; using type =
 04960
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<234»; };
                      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 =
04962
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<234»; }; // NOLINT template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
04963
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<173>, ZPZV<212>, ZPZV<153>, ZPZV<153>, ZPZV<7»; }; //
04964
                                     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
04965
                                      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>
04966
                                    template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type :
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<241, 12> { using ZPZ = aerobus::zpz<241>; using type =
04967
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<10>, ZPZV<109>, ZPZV<168>, ZPZV<22>,
                       ZPZV<197>, ZPZV<17>, ZPZV<7»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234»; }; // NOLINT
template<> struct ConwayPolynomial<241, 17> { using ZPZ = aerobus::zpz<241>; using type =
04969
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<234»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<241,
                                                                                                                                                                                                19> { using ZPZ = aerobus::zpz<241>; using type =
04970
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<234»; }; //</pre>
                       NOLINT
                                     template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
04971
                      POLYV<ZPZV<1>, ZPZV<245»; }; // NOLINT
                                      template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
                      POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; }; // NOLINT
04973
                                      template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT
template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<200>, ZPZV<6»; }; // NOLINT
04974
04975
                                       template<> struct ConwayPolynomial<251, 5> { using ZPZ = aerobus::zpz<251>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT
                     template<> struct ConwayPolynomial<251, 6> { using ZPZ = aerobus::zpz<251>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<247>, ZPZV<151>, ZPZV<179>, ZPZV<6»; }; // NOLINT
template<> struct ConwayPolynomial<251, 7> { using ZPZ = aerobus::zpz<251>; using type =
04976
04977
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<45»; };
                                      template<> struct ConwayPolynomial<251, 8> { using ZPZ = aerobus::zpz<251>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<142>, ZPZV<215>, ZPZV<173>, ZPZV<6»; }; //
04979
                                    template<> struct ConwayPolynomial<251, 9> { using ZPZ = aerobus::zpz<251>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<1062, ZPZV<245»;
                       }; // NOLINT
                                        template<> struct ConwayPolynomial<251, 10> { using ZPZ = aerobus::zpz<251>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<138>, ZPZV<110>, ZPZV<45>, ZPZV<34>,
                        ZPZV<149>, ZPZV<6»; }; // NOLINT</pre>
04981
                                      template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<26>, ZPZV<245»; }; // NOLINT
   template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<192>, ZPZV<53>, ZPZV<20>, ZPZV<20>, ZPZV<15>,
                       \text{ZPZV} < 201 >, \text{ZPZV} < 232 >, \text{ZPZV} < 6 >; }; // NOLINT
04983
                                       template<> struct ConwayPolynomial<251, 13> { using ZPZ = aerobus::zpz<251>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      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»; };</pre>
                      template<> struct ConwayPolynomial<251, 19> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04985
                      NOLINT
04986
                                       template<> struct ConwayPolynomial<257, 1> { using ZPZ = aerobus::zpz<257>; using type =
                      POLYV<ZPZV<1>, ZPZV<254»; }; // NOLINT
04987
                                    template<> struct ConwayPolynomial<257, 2> { using ZPZ = aerobus::zpz<257>; using type =
                      POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
04988
                      POLYV<ZPZV<1>, ZPZV<6>, 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
04990
                                     template<> struct ConwayPolynomial<257, 5> { using ZPZ = aerobus::zpz<257>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<254»; }; // NOLINT
                                      template<> struct ConwayPolynomial<257, 6> { using ZPZ = aerobus::zpz<257>; using type =
04991
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<62>, ZPZV<18>, ZPZV<138>, ZPZV<3»; }; // NOLINT
                                      template<> struct ConwayPolynomial<257,
04992
                                                                                                                                                                                                7> { using ZPZ = aerobus::zpz<257>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<254»; }; // NOL template<> struct ConwayPolynomial<257, 8> { using ZPZ = aerobus::zpz<257>; using type =
04993
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<179>, ZPZV<140>, ZPZV<162>, ZPZV<3»; }; //
                      NOLINT
04994
                                     template<> struct ConwayPolynomial<257, 9> { using ZPZ = aerobus::zpz<257>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<201>, ZPZV<50>, ZPZV<254»;
                   }; // NOLINT
                               template<> struct ConwayPolynomial<257, 10> { using ZPZ = aerobus::zpz<257>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<97>, ZPZV<12>, ZPZV<225>, ZPZV<180>, ZPZV<20>,
                   ZPZV<3»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<257, 11> { using ZPZ = aerobus::zpz<257>; using type =
04996
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<40>, ZPZV<254»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<257, 12> { using ZPZ = aerobus::zpz<257>; using type =
04997
                   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 =
04998
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<254»; };</pre>
                                                                                                                                                       // NOLINT
04999
                              template<> struct ConwayPolynomial<257, 17> { using ZPZ = aerobus::zpz<257>; using type
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, zPZV<0>, ZP
05000
                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<254»; }; //</pre>
05001
                                template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
                   POLYV<ZPZV<1>, ZPZV<258»; }; // NOLINT
                               template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
05002
                   POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; }; // NOLINT
                                template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; }; // NOLINT template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
05004
                   template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
05005
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<258»; }; // NOLINT
05006
                                 template<> struct ConwayPolynomial<263, 6> { using ZPZ = aerobus::zpz<263>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<250>, ZPZV<225>, ZPZV<25»; }; // NOLINT
05007
                              template<> struct ConwayPolynomial<263, 7> { using ZPZ = aerobus::zpz<263>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<258»; }; // NOLINT template<> struct ConwayPolynomial<263, 8> { using ZPZ = aerobus::zpz<263>; using type =
05008
                   POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<227>, ZPZV<170>, ZPZV<7>, ZPZV<5»; }; //
                                template<> struct ConwayPolynomial<263, 9> { using ZPZ = aerobus::zpz<263>; using type
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<261>, ZPZV<29>, ZPZV<258»;
                   }; // NOLINT
                   template<> struct ConwayPolynomial<263, 10> { using ZPZ = aerobus::zpz<263>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<245>, ZPZV<231>, ZPZV<198>, ZPZV<145>,
05010
                   ZPZV<119>, ZPZV<5»; };</pre>
                                                                                                // NOLINT
                                template<> struct ConwayPolynomial<263, 11> { using ZPZ = aerobus::zpz<263>; using type
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<2>, ZPZV<258»; }; // NOLINT</pre>
                   template<> struct ConwayPolynomial<263, 12> { using ZPZ = aerobus::zpz<263>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<174>, ZPZV<174>, ZPZV<162>, ZPZV<252>,
05012
                   ZPZV<47>, ZPZV<45>, ZPZV<180>, ZPZV<5»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
                   POLYV<ZPZV<1>, ZPZV<267»; }; // NOLINT
05014
                                 template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
                   POLYV<ZPZV<1>, ZPZV<268>, ZPZV<20»; }; // NOLINT template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
05015
                   POLYY<ZPZY<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 =
05017
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<267»; }; // NOLINT
05018
                               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
05019
                                 template<> struct ConwayPolynomial<269, 7> { using ZPZ = aerobus::zpz<269>; using type
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<66>, ZPZV<267»; }; //
05020
                               template<> struct ConwayPolynomial<269, 8> { using ZPZ = aerobus::zpz<269>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<220>, ZPZV<131>, ZPZV<232>, ZPZV<23; }; //
                   NOLINT
                               template<> struct ConwayPolynomial<269, 9> { using ZPZ = aerobus::zpz<269>; using type =
05021
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2), ZPZV<214>, ZPZV<267>, ZPZV<267>;
                   }; // NOLINT
                               template<> struct ConwayPolynomial<269, 10> { using ZPZ = aerobus::zpz<269>; using type =
05022
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<264>, ZPZV<243>, ZPZV<186>, ZPZV<61>,
                   ZPZV<10>, ZPZV<2»; }; // NOLINT
   template<> struct ConwayPolynomial<269, 11> { using ZPZ = aerobus::zpz<269>; using type :
05023
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<165>, ZPZV<63>, ZPZV<215>, ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2»; }; // NOLINT
05025
                               template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type =
                   POLYV<ZPZV<1>, ZPZV<265»; }; // NOLINT
                                template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
                  POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; }; // NOLINT
05027
                              template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6»; }; // NOLINT
05028
```

```
05029
                      template<> struct ConwayPolynomial<271, 5> { using ZPZ = aerobus::zpz<271>; using type =
             POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<2>, ZPZY<265»; }; // NOLINT template<> struct ConwayPolynomial<271, 6> { using ZPZ = aerobus::zpz<271>; using type =
05030
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<207>, ZPZV<207>, ZPZV<81>, ZPZV<66; }; // NOLINT template<> struct ConwayPolynomial<271, 7> { using ZPZ = aerobus::zpz<271>; using type =
05031
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; };
                                                                                                                                                                                                      // NOLINT
                     template<> struct ConwayPolynomial<271, 8> { using ZPZ = aerobus::zpz<271>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<114>, ZPZV<69>, ZPZV<6»; };
05033
                     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<0>, ZPZV<266>, ZPZV<186>, ZPZV<268»;
             }; // NOLINT
                      template<> struct ConwayPolynomial<271, 10> { using ZPZ = aerobus::zpz<271>; using type =
05034
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<13>, ZPZV<10>, ZPZV<26>, ZPZV<74>,
              ZPZV<126>, ZPZV<6»; }; // NOLINT</pre>
05035
                      template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
             ZPZV<10>, ZPZV<265»; j; // NOLINT template<> struct ConwayPolynomial<271, 12> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162>, ZPZV<210>, ZPZV<116>, ZPZV<205>,
             ZPZV<237>, ZPZV<256>, ZPZV<130>, ZPZV<6»; }; // NOLINT</pre>
05037
                      template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
             POLYV<ZPZV<1>, ZPZV<272»; }; // NOLINT
                     template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
05038
             POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; }; // NOLINT
                      template<> struct ConwayPolynomial277, 3> { using ZPZ = aerobus::zpz<277>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; }; // NOLINT template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
05040
             template<> struct ConwayPolynomial<277, 5> { using ZPZ = aerobus::zpz<277>; using type =
05041
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<272»; }; // NOLINT
05042
                      template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<9>, ZPZV<118>, ZPZV<5»; }; // NOLINT
05043
                    template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<272»; }; // NOLINT template<> struct ConwayPolynomial<277, 8> { using ZPZ = aerobus::zpz<277>; using type =
05044
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<159>, ZPZV<176>, ZPZV<5»; }; //
             NOLINT
                      template<> struct ConwayPolynomial<277, 9> { using ZPZ = aerobus::zpz<277>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<177>, ZPZV<110>, ZPZV<272»;
             }; // NOLINT
             template<> struct ConwayPolynomial<277, 10> { using ZPZ = aerobus::zpz<277>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<206>, ZPZV<253>, ZPZV<237>, ZPZV<241>,
05046
             ZPZV<260>, ZPZV<5»; };</pre>
                                                                 // NOLINT
                      template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
             ZPZV<5>, ZPZV<272»; }; // NOLINT</pre>
             template<> struct ConwayPolynomial<277, 12> { using ZPZ = aerobus::zpz<277>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<183>, ZPZV<218>, ZPZV<240>, ZPZV<40>, ZPZV<40>, ZPZV<180>, ZPZV<115>, ZPZV<202>, ZPZV<5»; }; // NOLINT
05048
                      template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
             POLYV<ZPZV<1>, ZPZV<278»; }; // NOLINT
05050
                      template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
             POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
05051
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<278»; }; // NOLINT
                      template<> struct ConwayPolynomial2281, 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 =
05053
             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 =
05054
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<151>, ZPZV<27>, ZPZV<27>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<281, 7> { using ZPZ = aerobus::zpz<281>; using type
05055
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<278»; }; //
05056
                     template<> struct ConwayPolynomial<281, 8> { using ZPZ = aerobus::zpz<281>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195>, ZPZV<279>, ZPZV<140>, ZPZV<3»; }; //
             NOLINT
                     template<> struct ConwayPolynomial<281, 9> { using ZPZ = aerobus::zpz<281>; using type =
05057
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<148>, ZPZV<70>, ZPZV<278»;
             }; // NOLINT
                     template<> struct ConwayPolynomial<281, 10> { using ZPZ = aerobus::zpz<281>; using type =
05058
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<145>, ZPZV<13>, ZPZV<138>,
             ZPZV<191>, ZPZV<3»; }; // NOLINT</pre>
                      template<> struct ConwayPolynomial<281, 11> { using ZPZ = aerobus::zpz<281>; using type =
05059
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<202>, ZPZV<68>, ZPZV<103>, ZPŽV<116>,
             ZPZV<58>, ZPZV<28>, ZPZV<191>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
05061
             POLYV<ZPZV<1>, ZPZV<280»; }; // NOLINT
05062
                      template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
             POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3»; }; // NOLINT
05063
                     template<> struct ConwayPolynomial<283, 3> { using ZPZ = aerobus::zpz<283>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; }; // NOLINT
05064
```

```
05065
                      template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
                    template<> struct ConwayPolynomial<283, 6> { using ZPZ = aerobus::zpz<283>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<68>, ZPZV<73>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type
05067
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<280»; };
                                                                                                                                                                                              // NOLTNT
                     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<23»; }; //
05069
                    template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<280»;
             }; // NOLINT
05070
                      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>, ZPŽV<100>,
             ZPZV<219>, ZPZV<3»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<283, 11> { using ZPZ = aerobus::zpz<283>; using type =
05071
             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<40>, ZPZV<20>, ZPZV<8>, ZPZV<96>, ZPZV<229>, ZPZV<49>,
             ZPZV<14>, ZPZV<56>, ZPZV<3»; }; // NOLINT</pre>
05073
                      template<> struct ConwayPolynomial<293, 1> { using ZPZ = aerobus::zpz<293>; using type =
            POLYV<ZPZV<1>, ZPZV<291»; }; // NOLINT
                     template<> struct ConwayPolynomial<293, 2> { using ZPZ = aerobus::zpz<293>; using type =
05074
            POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; }; // NOLINT
                      template<> struct ConwayPolynomial<293, 3> { using ZPZ = aerobus::zpz<293>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT template<> struct ConwayPolynomial<293, 4> { using ZPZ = aerobus::zpz<293>; using type =
05076
            template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
05077
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
05078
                      template<> struct ConwayPolynomial<293, 6> { using ZPZ = aerobus::zpz<293>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<210>, ZPZV<260>, ZPZV<2°, }; // NOLINT
05079
                    template<> struct ConwayPolynomial<293, 7> { using ZPZ = aerobus::zpz<293>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<291»; }; // NOLINT template<> struct ConwayPolynomial<293, 8> { using ZPZ = aerobus::zpz<293>; using type =
05080
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<2»; }; //
05081
                     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<0>, ZPZV<208>, ZPZV<208>, ZPZV<291»;
             }; // NOLINT
            template<> struct ConwayPolynomial<293, 10> { using ZPZ = aerobus::zpz<293>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<186>, ZPZV<28>, ZPZV<46>, ZPZV<184>, ZPZV<24>,
05082
             ZPZV<2»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<293, 11> { using ZPZ = aerobus::zpz<293>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
             ZPZV<3>, ZPZV<291»; }; // NOLINT</pre>
            template<> struct ConwayPolynomial<293, 12> { using ZPZ = aerobus::zpz<293>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<159>, ZPZV<210>, ZPZV<125>, ZPZV<212>, ZPZV<167>, ZPZV<144>, ZPZV<157>, ZPZV<22»; }; // NOLINT
05084
05085
                     template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
             POLYV<ZPZV<1>, ZPZV<302»; }; // NOLINT
05086
                     template<> struct ConwayPolynomial<307, 2> { using ZPZ = aerobus::zpz<307>; using type =
            POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
05087
            POLYY<ZPZY<1>, ZPZV<0>, ZPZV<7>, ZPZV<302»; }; // NOLINT template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5»; }; // NOLINT
                      template<> struct ConwayPolynomial<307, 5> { using ZPZ = aerobus::zpz<307>; using type =
05089
            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 =
05090
            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
05091
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
05092
                    template<> struct ConwayPolynomial<307, 8> { using ZPZ = aerobus::zpz<307>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<232>, ZPZV<131>, ZPZV<5»; }; //
             NOLINT
05093
                    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<70>, ZPZV<302»;
             }; // NOLINT
                     template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
05094
            POLYV<ZPZV<1>, ZPZV<294»; }; // NOLINT
                     template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
05095
            POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; }; // NOLINT
            template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; }; // NOLINT
05096
                    template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
05097
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
05098
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294»; }; // NOLINT
                     template<> struct ConwayPolynomial<311, 6> { using ZPZ = aerobus::zpz<311>; using type =
05099
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<167>, ZPZV<152>, ZPZV<17»; }; // NOLINT
                     template<> struct ConwayPolynomial<311, 7> { using ZPZ = aerobus::zpz<311>; using type
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<294»; };
05101
                    template<> struct ConwayPolynomial<311, 8> { using ZPZ = aerobus::zpz<311>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<162>, ZPZV<118>, ZPZV<2>, ZPZV<17»; }; //
             NOLTNT
```

```
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<24>, ZPZV<294»;
           }; // NOLINT
05103
                  template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
           POLYV<ZPZV<1>, ZPZV<303»; }; // NOLINT
                  template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
05104
           POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; }; // NOLINT
                   template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
05105
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; }; // NOLINT template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
05106
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<239>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
05107
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<303»; }; // NOLINT template<> struct ConwayPolynomial<313, 6> { using ZPZ = aerobus::zpz<313>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<196>, ZPZV<213>, ZPZV<253>, ZPZV<10»; }; // NOLINT
05109
                  template<> struct ConwayPolynomial<313, 7> { using ZPZ = aerobus::zpz<313>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05110
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<306>, ZPZV<99>, ZPZV<106>, ZPZV<10*, }; //
           template<> struct ConwayPolynomial<313, 9> { using ZPZ = aerobus::zpz<313>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26
05111
           }; // NOLINT
                  template<> struct ConwayPolynomial<317, 1> { using ZPZ = aerobus::zpz<317>; using type =
05112
           POLYV<ZPZV<1>, ZPZV<315»; // NOLINT
                   template<> struct ConwayPolynomial<317, 2> { using ZPZ = aerobus::zpz<317>; using type =
           POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; }; // NOLINT
05114
                  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 =
05115
05116
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<315»; // NOLINT
                 template<> struct ConwayPolynomial<317, 6> { using ZPZ = aerobus::zpz<317>; using type =
0.5117
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<195>, ZPZV<156>, ZPZV<4>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<317, 7> { using ZPZ = aerobus::zpz<317>; using type =
05118
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<315»; }; // NOLINT template<> struct ConwayPolynomial<317, 8> { using ZPZ = aerobus::zpz<317>; using type =
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<207>, ZPZV<85>, ZPZV<31>, ZPZV<2w; };
05120
                 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<284>, ZPZV<296>, ZPZV<315»;
           }; // NOLINT
05121
                   template<> struct ConwayPolynomial<331, 1> { using ZPZ = aerobus::zpz<331>; using type =
           POLYV<ZPZV<1>, ZPZV<328»; }; // NOLINT
05122
                   template<> struct ConwayPolynomial<331, 2> { using ZPZ = aerobus::zpz<331>; using type =
           POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; }; // NOLINT
05123
                  template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<328»; }; // NOLINT
                  template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
05124
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<290>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<331, 5> { using ZPZ = aerobus::zpz<331>; using type =
05125
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<328»; }; // NOLINT
05126
                  template<> struct ConwayPolynomial<331, 6> { using ZPZ = aerobus::zpz<331>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<255>, ZPZV<159>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<331, 7> { using ZPZ = aerobus::zpz<331>; using type
05127
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<328»; }; //
                 template<> struct ConwayPolynomial<331, 8> { using ZPZ = aerobus::zpz<331>; using type =
05128
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<24>, ZPZV<308>, ZPZV<78>, ZPZV<38; };
           NOLINT
                  template<> struct ConwayPolynomial<331, 9> { using ZPZ = aerobus::zpz<331>; using type =
05129
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<194>, ZPZV<210>, ZPZV<328»;
           }; // NOLINT
                   template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
           POLYV<ZPZV<1>, ZPZV<327»; }; // NOLINT
05131
                  template<> struct ConwayPolynomial<337, 2> { using ZPZ = aerobus::zpz<337>; using type =
           POLYV<ZPZV<1>, ZPZV<332>, ZPZV<10»; }; // NOLINT
          template<> struct ConwayPolynomial<337, 3> { using ZPZ = aerobus::zpz<337>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT
05132
                   template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
05133
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10»; }; // NOLINT
05134
                 template<> struct ConwayPolynomial<337, 5> { using ZPZ = aerobus::zpz<337>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT template<> struct ConwayPolynomial<337, 6> { using ZPZ = aerobus::zpz<337>; using type =
05135
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<216>, ZPZV<109>, ZPZV<109>, ZPZV<109>; }; // NOLINT template<> struct ConwayPolynomial<337, 7> { using ZPZ = aerobus::zpz<337>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<327»; }; // NOLINT
05137
                  template<> struct ConwayPolynomial<337, 8> { using ZPZ = aerobus::zpz<337>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<331>, ZPZV<246>, ZPZV<251>, ZPZV<10»; }; //
           NOLINT
                  template<> struct ConwayPolynomial<337, 9> { using ZPZ = aerobus::zpz<337>; using type =
05138
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<148>, ZPZV<98>, ZPZV<327»;
           }; // NOLINT
05139
                  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 =
0.5140
           POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
      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 template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
0.5143
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<345»; }; // NOLINT
           template<> struct ConwayPolynomial<347, 6> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<343>, ZPZV<26>, ZPZV<56>, ZPZV<2»; };
          template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
05145
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type =
05146
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<187>, ZPZV<213>, ZPZV<117>, ZPZV<2*; }; //
           template<> struct ConwayPolynomial<347, 9> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<235>, ZPZV<252>, ZPZV<252>, ZPZV<345»;
      }; // NOLINT
05148
           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 =
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; }; // NOLINT template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
05151
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2*; ); // NOLINT template<> struct ConwayPolynomial<349, 5> { using ZPZ = aerobus::zpz<349>; using type =
05152
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<347»; }; // NOLINT
           template<> struct ConwayPolynomial<349, 6> { using ZPZ = aerobus::zpz<349>; using type =
05153
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<135>, ZPZV<177>, ZPZV<316>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<349, 7> { using ZPZ = aerobus::zpz<349>; using type
05154
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<10>, ZPZV<347»; }; // NOLINT
          template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type =
05155
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<328>, ZPZV<268>, ZPZV<268>, ZPZV<268; }; //
          template<> struct ConwayPolynomial<349, 9> { using ZPZ = aerobus::zpz<349>; using type =
05156
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<290>, ZPZV<130>, ZPZV<347»;
      }; // NOLINT
      template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type = POLYV<ZPZV<1>, ZPZV<350»; }; // NOLINT
05157
           template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3»; }; // NOLINT
05159
          template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<3»; }; // NOLINT
0.5160
           template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<350»; }; // NOLINT
05162
          template<> struct ConwayPolynomial<353, 6> { using ZPZ = aerobus::zpz<353>; using type =
      template<> struct ConwayPolynomial<353, 7> { using ZPZ = aerobus::zpz<353>; using type
05163
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<350»; }; // NOLINT
           template<> struct ConwayPolynomial<353, 8> { using ZPZ = aerobus::zpz<353>; using type
05164
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<182>, ZPZV<26>, ZPZV<37>, ZPZV<37>, ZPZV<39; };
      NOLINT
      05165
      }; // NOLINT
           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 =
05167
      POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7»; }; // NOLINT
          template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
05168
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<352»; }; // NOLINT template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
05169
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<229>, ZPZV<7»; }; // NOLINT
05170
          template<> struct ConwayPolynomial<359, 5> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; }; // NOLINT
05171
           template<> struct ConwayPolynomial<359, 6> { using ZPZ = aerobus::zpz<359>; using type =
      POLYY-ZPZV-1>, ZPZV-4>, ZPZV-4>, ZPZV-309>, ZPZV-327>, ZPZV-327>, ZPZV-7»; }; // NOLINT template<> struct ConwayPolynomial<359, 7> { using ZPZ = aerobus::zpz-359>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; };
           template<> struct ConwayPolynomial<359, 8> { using ZPZ = aerobus::zpz<359>; using type
05173
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<301>, ZPZV<143>, ZPZV<271>, ZPZV<7»; }; //
      NOLINT
05174
           template<> struct ConwayPolynomial<359, 9> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<356>, ZPZV<165>, ZPZV<352»;
      }; // NOLINT
05175
           template<> struct ConwayPolynomial<367, 1> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<361»; }; // NOLINT
           template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
05176
      POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6»; }; // NOLINT
           template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
05177
      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
05179
          template<> struct ConwayPolynomial<367, 5> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<361»; }; // NOLINT
05180
          template<> struct ConwayPolynomial<367, 6> { using ZPZ = aerobus::zpz<367>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<222>, ZPZV<321>, ZPZV<324>, ZPZV<6»; };
            template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<361»; };
          template<> struct ConwayPolynomial<367, 8> { using ZPZ = aerobus::zpz<367>; using type =
05182
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<335>, ZPZV<282>, ZPZV<50>, ZPZV<6»; }; //
      NOT.TNT
      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<213>, ZPZV<213>, ZPZV<268>, ZPZV<361»;
       }; // NOLINT
05184
           template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<371»; }; // NOLINT
           template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
05185
      POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<371»; }; // NOLINT
           template<> struct ConwayPolynomial<373, 4> { using ZPZ = aerobus::zpz<373>; using type =
05187
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<373, 5> { using ZPZ = aerobus::zpz<373>; using type =
05188
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<371»; }; // NOLINT
           template<> struct ConwayPolynomial<373, 6> { using ZPZ = aerobus::zpz<373>; using type =
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<1>, ZPZV<126>, ZPZV<83>, ZPZV<108>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type =
05190
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<371»; }; // NOLINT template<> struct ConwayPolynomial<373, 8> { using ZPZ = aerobus::zpz<373>; using type =
0.5191
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<203>, ZPZV<219>, ZPZV<66>, ZPZV<2»; };
           template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type =
05192
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<238>, ZPZV<370>, ZPZV<371»;
       }; // NOLINT
05193
           template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<377»; }; // NOLINT
05194
            template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
       POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2»; }; // NOLINT
05195
          template<> struct ConwayPolynomial<379, 3> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377»; ); // NOLINT
template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<32>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<379, 5> { using ZPZ = aerobus::zpz<379>; using type =
05196
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<377»; }; // NOLINT
           template<> struct ConwayPolynomial<379, 6> { using ZPZ = aerobus::zpz<379>; using type =
05198
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<374>, ZPZV<364>, ZPZV<246>, ZPZV<2*; }; // NOLINT
05199
           template<> struct ConwayPolynomial<379, 7> { using ZPZ = aerobus::zpz<379>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<377»; }; // NOLINT
           template<> struct ConwayPolynomial<379, 8> { using ZPZ = aerobus::zpz<379>; using type =
05200
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<210>, ZPZV<194>, ZPZV<173>, ZPZV<2»; }; //
       NOLINT
05201
           template<> struct ConwayPolynomial<379, 9> { using ZPZ = aerobus::zpz<379>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<362>, ZPZV<369>, ZPZV<377»;
       }; // NOLINT
05202
            template<> struct ConwavPolynomial<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 =
       POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
05204
      POLYV<ZPZV<1>, ZPZV<0, ZPZV<1>, ZPZV<378»; }; // NOLINT
template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5»; }; // NOLINT
05205
           template<> struct ConwayPolynomial<383, 5> { using ZPZ = aerobus::zpz<383>; using type =
05206
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
05207
            template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<8>, ZPZV<158>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 7> { using ZPZ = aerobus::zpz<383>; using type
05208
      POLYV<ZPZV<1>, 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
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<281>, ZPZV<332>, ZPZV<296>, ZPZV<5»; }; //
       NOLINT
05210
      template<> struct ConwayPolynomial<383, 9> { using ZPZ = aerobus::zpz<383>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137>, ZPZV<137>, ZPZV<137>, ZPZV<137>, ZPZV<137</pre>
      }; // NOLINT
05211
            template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
       POLYV<ZPZV<1>, ZPZV<387»; }; // NOLINT
05212
           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 =
05213
      POLYY<ZPZY<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
05215
            template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<387»; }; // NOLINT
05216
           template<> struct ConwayPolynomial<389, 6> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<339>, ZPZV<255>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<389,
                                                          7> { using ZPZ = aerobus::zpz<389>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24, ZPZV<387»; }; // NOL template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::zpz<389>; using type
05218
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<351>, ZPZV<19>, ZPZV<290>, ZPZV<2»; };
      NOLINT
05219
           template<> struct ConwayPolynomial<389, 9> { using ZPZ = aerobus::zpz<389>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<308>, ZPZV<387»;
05220
             template<> struct ConwayPolynomial<397, 1> { using ZPZ = aerobus::zpz<397>; using type =
        POLYV<ZPZV<1>, ZPZV<392»; }; // NOLINT
05221
              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 =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<392»; }; // NOLINT
             template<> struct ConwayPolynomial<397, 4> { using ZPZ = aerobus::zpz<397>; using type =
05223
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
05224
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392»; }; // NOLINT
05225
              template<> struct ConwayPolynomial<397, 6> { using ZPZ = aerobus::zpz<397>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<274>, ZPZV<287>, ZPZV<5»; }; // NOLIN
05226
             template<> struct ConwayPolynomial<397, 7> { using ZPZ = aerobus::zpz<397>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<392»; }; // NOLT template<> struct ConwayPolynomial<397, 8> { using ZPZ = aerobus::zpz<397>; using type =
05227
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<375>, ZPZV<255>, ZPZV<203>, ZPZV<5»; }; //
             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<252>, ZPZV<392»;
        }; // NOLINT
05229
             template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
        POLYV<ZPZV<1>, ZPZV<398»; }; // NOLINT
05230
              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 =
05231
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<398»; }; // NOLINT template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type =
05232
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<372>, ZPZV<3»; }; // NOLINT

template<> struct ConwayPolynomial<401, 5> { using ZPZ = aerobus::zpz<401>; using type =
05233
        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 =
05234
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<115>, ZPZV<81>, ZPZV<51>, ZPZV<3»; }; // NOLINT
05235
              template<> struct ConwayPolynomial<401, 7> { using ZPZ = aerobus::zpz<401>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<398»; }; // NOLINT
             template<> struct ConwayPolynomial<401, 8> { using ZPZ = aerobus::zpz<401>; using type =
05236
        POLYV<ZPZV<1>, ZPZV<0>, ZPŽV<0>, ZPZV<0>, ZPZV<0>, ZPZV<380>, ZPZV<113>, ZPZV<164>, ZPŽV<3»; }; //
             template<> struct ConwayPolynomial<401, 9> { using ZPZ = aerobus::zpz<401>; using type =
05237
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<158>, ZPZV<398»;
        }; // NOLINT
              template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
05238
        POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
        POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT
05240
             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 =
05241
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<12>, ZPZV<407>, ZPZV<407>,
05242
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<388»; }; // NOLINT
05243
              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 template<> struct ConwayPolynomial<409, 7> { using ZPZ = aerobus::zpz<409>; using type :
05244
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<388»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<256>, ZPZV<69>, ZPZV<396>, ZPZV<21»; }; //
05246
              template<> struct ConwayPolynomial<409, 9> { using ZPZ = aerobus::zpz<409>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<318>, ZPZV<318>, ZPZV<318»;
        }; // NOLINT
05247
              template<> struct ConwayPolynomial<419, 1> { using ZPZ = aerobus::zpz<419>; using type =
        POLYV<ZPZV<1>, ZPZV<417»; }; // NOLINT
05248
             template<> struct ConwayPolynomial<419, 2> { using ZPZ = aerobus::zpz<419>; using type =
        POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT
05249
              template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<11>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type =
05250
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<373>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<419, 5> { using ZPZ = aerobus::zpz<419>; using type =
05251
        05252
              template<> struct ConwayPolynomial<419, 6> { using ZPZ = aerobus::zpz<419>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<411>, ZPZV<33>, ZPZV<257>, ZPZV<2»; }; // NOLINT
                                                                      7> { using ZPZ = aerobus::zpz<419>; using type
05253
              template<> struct ConwayPolynomial<419,
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; //
             template<> struct ConwayPolynomial<419, 8> { using ZPZ = aerobus::zpz<419>; using type
05254
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<234>, ZPZV<388>, ZPZV<151>, ZPZV<2»; }; //
        NOLINT
        template<> struct ConwayPolynomial<419, 9> { using ZPZ = aerobus::zpz<419>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3417»;</pre>
05255
        }; // NOLINT
              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 =
05257
        POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT
             template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
05258
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419»; };
           template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<421, 5> { using ZPZ = aerobus::zpz<421>; using type =
05260
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<419»; }; // NOLINT
      template<> struct ConwayPolynomial<421, 6> { using ZPZ = aerobus::zpz<421>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<111>, ZPZV<342>, ZPZV<41>, ZPZV<2»; }; // NOLINT
05261
           template<> struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type
05262
      POLYV<ZPZV<1>, ZPZV<0>, 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 =
05263
      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 =
05264
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1*>, ZPZV<145>, ZPZV<4145>, ZPZV<419*;
      }; // NOLINT
05265
           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 =
05266
      POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<431, 3> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424»; }; // NOLINT template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
05268
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<431, 5> { using ZPZ = aerobus::zpz<431>; using type =
05269
      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 =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<161>, ZPZV<202>, ZPZV<182>, ZPZV<7»; }; // NOLINI
05271
           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<424»; }; // NOLINT
05272
           template<> struct ConwayPolynomial<431, 8> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, 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 =
05273
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<71>, ZPZV<329>, ZPZV<424%;
      }; // NOLINT
05274
           template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<428»; }; // NOLINT
           template<> struct ConwayPolynomial<433, 2> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<433, 3> { using ZPZ = aerobus::zpz<433>; using type =
05276
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428»; }; // NOLINT template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
05277
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<402>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<433, 5> { using ZPZ = aerobus::zpz<433>; using type =
05278
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<428»; }; // NOLINT
05279
           template<> struct ConwayPolynomial<433, 6> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<244>, ZPZV<353>, ZPZV<360>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<433, 7> { using ZPZ = aerobus::zpz<433>; using type =
05280
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<428»; }; // NOLINT
           template<> struct ConwayPolynomial<433, 8> { using ZPZ = aerobus::zpz<433>; using type =
05281
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347>, ZPZV<32>, ZPZV<39>, ZPZV<55; }; //
05282
           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<448»;
      }; // NOLINT
05283
           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 =
05284
      POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
05285
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424»; ); // NOLINT
template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; }; // NOLINT
05286
           template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT
05288
           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
05289
           template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424w; }; // NOLINT
           template<> struct ConwayPolynomial<439, 8> { using ZPZ = aerobus::zpz<439>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<296>, ZPZV<266>, ZPZV<25»; }; //
      05291
      }; // NOLINT
           template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<441»; }; // NOLINT
05293
           template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
05294
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41»; }; // NOLINT template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<4443>; using type =
05296
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT template<> struct ConwayPolynomial<443, 6> { using ZPZ = aerobus::zpz<443>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<298>, ZPZV<218>, ZPZV<41>, ZPZV<2»; }; // NOLINT
05297
```

```
template<> struct ConwayPolynomial<443, 7> { using ZPZ = aerobus::zpz<443>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<641»; }; // NOLINT
             template<> struct ConwayPolynomial<443, 8> { using ZPZ = aerobus::zpz<443>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<217>, ZPZV<290>, ZPZV<2»; }; //
        NOLINT
              template<> struct ConwayPolynomial<443, 9> { using ZPZ = aerobus::zpz<443>; using type =
05300
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<125>, ZPZV<109, ZPZV<441»;
        }; // NOLINT
              template<> struct ConwayPolynomial<449, 1> { using ZPZ = aerobus::zpz<449>; using type =
05301
        POLYV<ZPZV<1>, ZPZV<446»; }; // NOLINT
              template<> struct ConwayPolynomial<449, 2> { using ZPZ = aerobus::zpz<449>; using type =
05302
        POLYV<ZPZV<1>, ZPZV<444>, ZPZV<3»; }; // NOLINT
05303
              template<> struct ConwayPolynomial<449, 3> { using ZPZ = aerobus::zpz<449>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446»; }; // NOLINT
05304
             template<> struct ConwayPolynomial<449, 4> { using ZPZ = aerobus::zpz<449>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<449, 5> { using ZPZ = aerobus::zpz<449>; using type =
05305
        POLYV<ZPZV<1>, ZPZV<0>, 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»; };
              template<> struct ConwayPolynomial<449, 7> { using ZPZ = aerobus::zpz<449>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<28>, ZPZV<446»; };
05308
              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*; }; //
        NOLINT
              template<> struct ConwayPolynomial<449, 9> { using ZPZ = aerobus::zpz<449>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<26>, ZPZV<26>, ZPZV<29>, ZPZV<4446»; };
         // NOLINT
05310
              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 =
05311
        POLYV<ZPZV<1>, ZPZV<454>, ZPZV<13»; };
                                                                  // NOLINT
              template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
05312
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444\text{w}; }; // NOLINT
              template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
05313
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13»; }; // NOLINT
              template<> struct ConwayPolynomial<457, 5> { using ZPZ = aerobus::zpz<457>; using type =
05314
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<444»; }; // NOLINT
05315
              template<> struct ConwayPolynomial<457, 6> { using ZPZ = aerobus::zpz<457>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<205>, ZPZV<266>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<457, 7> { using ZPZ = aerobus::zpz<457>; using type =
05316
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444*; }; // NOLINT template<> struct ConwayPolynomial<457, 8> { using ZPZ = aerobus::zpz<457>; using type =
05317
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<365>, ZPZV<296>, ZPZV<412>, ZPZV<13»; }; //
05318
              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<444»;
        }; // NOLINT
05319
              template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<459»; }; // NOLINT
05320
              template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2»; }; // NOLINT
05321
              template<> struct ConwayPolynomial<461, 3> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459»; }; // NOLINT template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
05322
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<459»; }; // NOLINT
              template<> struct ConwayPolynomial<461, 6> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<439>, ZPZV<432>, ZPZV<329>, ZPZV<2»; }; // NOLINT
05325
              template<> struct ConwayPolynomial<461, 7> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<459»; }; // NOLINT
              template<> struct ConwayPolynomial<461, 8> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<449>, ZPZV<321>, ZPZV<2»; }; //
        template<> struct ConwayPolynomial<461, 9> { using ZPZ = aerobus::zpz<461>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<210>, ZPZV<216>, ZPZV<459»;
05327
        }; // NOLINT
              template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
05328
        POLYV<ZPZV<1>, ZPZV<460»; }; // NOLINT
              template<> struct ConwayPolynomial<463, 2> { using ZPZ = aerobus::zpz<463>; using type =
        POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
05330
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<460»; }; // NOLINT template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; }; // NOLINT
05331
              template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
05332
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<460»; }; // NOLINT
05333
              template<> struct ConwayPolynomial<463, 6> { using ZPZ = aerobus::zpz<463>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<462>, ZPZV<51>, ZPZV<110>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<463, 7> { using ZPZ = aerobus::zpz<463>; using type = DOLYVZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0 - Z
05334
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<13>, ZPZV<460»; }; // NOLINT
              template<> struct ConwayPolynomial<463, 8> { using ZPZ = aerobus::zpz<463>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3414>, ZPZV<396>, ZPZV<3»; }; //
        template<> struct ConwayPolynomial<463, 9> { using ZPZ = aerobus::zpz<463>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<433>, ZPZV<433>, ZPZV<227>, ZPZV<460»;</pre>
05336
```

```
}; // NOLINT
05337
             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 =
05338
       POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
05339
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
05340
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<467, 5> { using ZPZ = aerobus::zpz<467>; using type =
05341
       POLYV<ZPZV<1>, 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<123>, ZPZV<62>, ZPZV<237>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type =
05342
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<465»; }; // NOLINT
05344
            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<28; }; //
       NOLINT
            template<> struct ConwayPolynomial<467, 9> { using ZPZ = aerobus::zpz<467>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<397>, ZPZV<447>, ZPZV<465»;
       }; // NOLINT
05346
            template<> struct ConwayPolynomial<479, 1> { using ZPZ = aerobus::zpz<479>; using type =
       POLYV<ZPZV<1>, ZPZV<466»; }; // NOLINT
            template<> struct ConwayPolynomial<479, 2> { using ZPZ = aerobus::zpz<479>; using type =
05347
       POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; }; // NOLINT
            template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466»; }; // NOLINT
            template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
05349
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<479, 5> { using ZPZ = aerobus::zpz<479>; using type =
05350
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<466»; }; // NOLINT
05351
            template<> struct ConwayPolynomial<479, 6> { using ZPZ = aerobus::zpz<479>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<243>, ZPZV<287>, ZPZV<334>, ZPZV<13»; }; // NOLINT
05352
           template<> struct ConwayPolynomial<479, 7> { using ZPZ = aerobus::zpz<479>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<466»; }; // NOLINT template<> struct ConwayPolynomial<479, 8> { using ZPZ = aerobus::zpz<479>; using type =
05353
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<247>, ZPZV<440>, ZPZV<17>, ZPZV<13»; }; //
            template<> struct ConwayPolynomial<479, 9> { using ZPZ = aerobus::zpz<479>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<185>, ZPZV<466»; };
       // NOLINT
            template<> struct ConwayPolynomial<487, 1> { using ZPZ = aerobus::zpz<487>; using type =
05355
       POLYV<ZPZV<1>, ZPZV<484»; }; // NOLINT
            template<> struct ConwayPolynomial</br>
487, 2> { using ZPZ = aerobus::zpz<487>; using type =
05356
       POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; }; // NOLINT
05357
            template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; }; // NOLINT
template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
05358
05359
       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
       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<7>, ZPZV<484%; }; // NOLINT
05361
            template<> struct ConwayPolynomial<487, 8> { using ZPZ = aerobus::zpz<487>; using type =
05362
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<249>, ZPZV<137>, ZPZV<3»; }; //
       template<> struct ConwayPolynomial<487, 9> { using ZPZ = aerobus::zpz<487>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<271>, ZPZV<4447>, ZPZV<484»;
       }; // NOLINT
            template<> struct ConwayPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
05364
       POLYV<ZPZV<1>, ZPZV<489»; }; // NOLINT
            template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
05366
       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 =
05367
05368
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT
05369
           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<2x; }; // NOLINT template<> struct ConwayPolynomial<491, 7> { using ZPZ = aerobus::zpz<491>; using type :
05370
       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4, ZPZV<0>, ZPZV<4, ZPZV<5>, ZPZV<489»; }; // NOLIN template<> struct ConwayPolynomial<491, 8> { using ZPZ = aerobus::zpz<491>; using type =
                                                                                                                // NOLINT
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378>, ZPZV<372>, ZPZV<216>, ZPZV<2»; };
05372
            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<0>, ZPZV<0>, ZPZV<149>, ZPZV<453>, ZPZV<489»;
       }; // NOLINT
            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 =
       POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; }; // NOLINT
```

```
05376
           template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<4>, ZPZV<495>, ZPZV<495>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
05377
      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<7»; }; // NOLINT
05378
           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<0>, ZPZV<0>, ZPZV<492»; };
          template<> struct ConwayPolynomial<499, 8> { using ZPZ = aerobus::zpz<499>; using type =
05380
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<288>, ZPZV<309>, ZPZV<200>, ZPZV<7»; }; //
      NOLINT
          template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type =
05381
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<491>, ZPZV<222>, ZPZV<492»;
      }; // NOLINT
05382
           template<> struct ConwayPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<498»; }; // NOLINT
          05383
      POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
05384
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<498»; }; // NOLINT template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<325>, ZPZV<5»; }; // NOLINT
          template<> struct ConwayPolynomial<503, 5> { using ZPZ = aerobus::zpz<503>; using type =
05386
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<498»; }; // NOLINT
05387
           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 =
05388
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<498»; }; // NOLINT template<> struct ConwayPolynomial<503, 8> { using ZPZ = aerobus::zpz<503>; using type =
05389
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<441>, ZPZV<203>, ZPZV<316>, ZPZV<35»; }; //
      NOLINT
05390
           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
05391
           template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<507»; }; // NOLINT
           template<> struct ConwayPolynomial<509, 2> { using ZPZ = aerobus::zpz<509>; using type =
05392
      POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2»; }; // NOLINT
05393
           template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<507»; }; // NOLINT template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
05394
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
05395
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
           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
05397
          template<> struct ConwayPolynomial<509, 7> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<507»; }; // NOLINT
05398
          template<> struct ConwayPolynomial<509, 8> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<473>, ZPZV<382>, ZPZV<2»; }; //
           template<> struct ConwayPolynomial<509, 9> { using ZPZ = aerobus::zpz<509>; using type =
05399
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<314>, ZPZV<28>, ZPZV<507»;
      }; // NOLINT
05400
           template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<518»; }; // NOLINT
           template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
05402
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518»; }; // NOLINT
template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<509>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<521, 5> { using ZPZ = aerobus::zpz<521>; using type =
05403
05404
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<518»; }; // NOLINT
05405
          template<> struct ConwayPolynomial<521, 6> { using ZPZ = aerobus::zpz<521>; using type =
      05406
           template<> struct ConwayPolynomial<521, 7> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<518»; }; // NOLINT
          template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type =
05407
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<462>, ZPZV<407>, ZPZV<312>, ZPZV<3»; };
05408
          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<5>, ZPZV<181>, ZPZV<483>, ZPZV<518»;
      }; // NOLINT
05409
           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 =
05410
      POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; }; // NOLINT
05411
           template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
05412
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2»; };
                                                                     // NOLINT
           template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<521»; }; // NOLINT
05414
          template<> struct ConwayPolynomial<523, 6> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<475>, ZPZV<475>, ZPZV<371>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<523, 7> { using ZPZ = aerobus::zpz<523>; using type =
05415
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<13>, ZPZV<13>, ZPZV<521»; };
                template<> struct ConwayPolynomial<523, 8> { using ZPZ = aerobus::zpz<523>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<518>, ZPZV<184>, ZPZV<380>, ZPZV<2»; }; //
         NOLINT
05417
                template<> struct ConwayPolynomial<523, 9> { using ZPZ = aerobus::zpz<523>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<342>, ZPZV<345>, ZPZV<521»;
05418
                template<> struct ConwayPolynomial<541, 1> { using ZPZ = aerobus::zpz<541>; using type =
         POLYV<ZPZV<1>, ZPZV<539»; }; // NOLINT
               template<> struct ConwayPolynomial<541, 2> { using ZPZ = aerobus::zpz<541>; using type =
05419
         POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; }; // NOLINT
                template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
05420
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<2>, ZPZV<539»; }; // NOLINT template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2»; }; // NOLINT
         template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<539»; }; // NOLINT</pre>
05422
         template<> struct ConwayPolynomial541, 6> { using ZPZ = aerobus::zpz<541>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<239>, ZPZV<320>, ZPZV<69>, ZPZV<2»; }; // NOLINT</pre>
05423
                template<> struct ConwayPolynomial<541,
                                                                               7> { using ZPZ = aerobus::zpz<541>; using type
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<539»; }; //
05425
                template<> struct ConwayPolynomial<541, 8> { using ZPZ = aerobus::zpz<541>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<376>, ZPZV<108>, ZPZV<113>, ZPZV<2»; }; //
         NOLINT
05426
                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
05427
               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 =
05428
         POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; }; // NOLINT
05429
                template<> struct ConwayPolynomial<547, 3> { using ZPZ = aerobus::zpz<547>; using type =
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<4>, ZPZY<45, ZPZY<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 4> { using ZPZ = aerobus::zpz<547>; using type =
05430
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<334>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<547, 5> { using ZPZ = aerobus::zpz<547>; using type =
05431
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<2>, ZPZY<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 6> { using ZPZ = aerobus::zpz<547>; using type =
         POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<547, 7> { using ZPZ = aerobus::zpz<547>; using type =
05433
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 8> { using ZPZ = aerobus::zpz<547>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2»; }; //
05434
         template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<263>, ZPZV<545»;</pre>
          }; // NOLINT
05436
                template<> struct ConwayPolynomial<557, 1> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<555»; }; // NOLINT
                template<> struct ConwayPolynomial<557, 2> { using ZPZ = aerobus::zpz<557>; using type =
05437
         POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; }; // NOLINT
                template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<555»; }; // NOLINT
template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; }; // NOLINT
05439
                template<> struct ConwayPolynomial<557, 5> { using ZPZ = aerobus::zpz<557>; using type =
05440
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<555»; }; // NOLINT
               template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type =
05441
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<202>, ZPZV<192>, ZPZV<253>, ZPZV<2; }; // NOLINT template<> struct ConwayPolynomial<557, 7> { using ZPZ = aerobus::zpz<557>; using type =
05442
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<555»; }; // NOLINT template<> struct ConwayPolynomial<557, 8> { using ZPZ = aerobus::zpz<557>; using type =
05443
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<384>, ZPZV<113>, ZPZV<2»; }; //
               template<> struct ConwayPolynomial<557, 9> { using ZPZ = aerobus::zpz<557>; using type =
05444
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<456>, ZPZV<434>, ZPZV<555»;
          }; // NOLINT
                \texttt{template<> struct ConwayPolynomial<563, 1> \{ using \ ZPZ = aerobus:: zpz<563>; \ using \ type = aerobus:: zpz<563>; \ 
05445
         POLYV<ZPZV<1>, ZPZV<561»; }; // NOLINT
05446
                template<> struct ConwayPolynomial<563, 2> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; }; // NOLINT
05447
               template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; }; // NOLINT template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
05448
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<20>, ZPZV<399>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<563, 5> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<561»; }; // NOLINT
05450
                template<> struct ConwayPolynomial<563, 6> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<122>, ZPZV<303>, ZPZV<246>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz<563>; using type =
05451
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<561»; }; // NOLINT
                template<> struct ConwayPolynomial<563, 8> { using ZPZ = aerobus::zpz<563>; using type =
05452
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<176>, ZPZV<509>, ZPZV<2»; }; //
         NOLINT
05453
               template<> struct ConwayPolynomial<563, 9> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<15>, ZPZV<19>, ZPZV<561»; };
          // NOLINT
```

```
05454
                  template<> struct ConwayPolynomial<569, 1> { using ZPZ = aerobus::zpz<569>; using type =
          POLYV<ZPZV<1>, ZPZV<566»; }; // NOLINT
05455
                 template<> struct ConwayPolynomial<569, 2> { using ZPZ = aerobus::zpz<569>; using type =
          POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; }; // NOLINT
05456
                 template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; }; // NOLINT
                 template<> struct ConwayPolynomial<569, 5> { using ZPZ = aerobus::zpz<569>; using type =
05458
           \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<4>, \verb"ZPZV<566"; \verb"]; $ // \verb"NOLINT" | NOLINT" 
                 template<> struct ConwayPolynomial<569, 6> { using ZPZ = aerobus::zpz<569>; using type =
05459
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<50>, ZPZV<263>, ZPZV<480>, ZPZV<3»; }; // NOLINT
05460
                 template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<566»; }; //
05461
                 template<> struct ConwayPolynomial<569, 8> { using ZPZ = aerobus::zpz<569>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<527>, ZPZV<173>, ZPZV<241>, ZPZV<241>, ZPZV<3»; }; //
          NOT.TNT
          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<478>, ZPZV<478>, ZPZV<566>, ZPZV<566>;
05462
                 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 =
05464
          POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
05465
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<6>, ZPZY<68», ZPZY<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type =
05466
          05467
                  template<> struct ConwayPolynomial<571, 5> { using ZPZ = aerobus::zpz<571>; using type =
          POLYV-ZPZV-1>, ZPZV-(>, ZPZV-(>), ZP
05468
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<221>, ZPZV<295>, ZPZV<33>, ZPZV<3»; };
                 template<> struct ConwayPolynomial<571, 7> { using ZPZ = aerobus::zpz<571>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<568»; }; // NOLINT
05470
                 template<> struct ConwayPolynomial<571, 8> { using ZPZ = aerobus::zpz<571>; using type =
          POLYV<2PZV<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<34>, ZPZV<545>, ZPZV<179>, ZPZV<568»;
          }; // NOLINT
05472
                 template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
          POLYV<ZPZV<1>, ZPZV<572»; }; // NOLINT
                 template<> struct ConwayPolynomial<577, 2> { using ZPZ = aerobus::zpz<577>; using type =
05473
          POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5»; }; // NOLINT
                  template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572»; };
                                                                                                // NOLINT
05475
                 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 =
05476
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<572»; }; // NOLINT
05477
                  template<> struct ConwayPolynomial<577, 6> { using ZPZ = aerobus::zpz<577>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<450>, ZPZV<25>, ZPZV<283>, ZPZV<5»; }; // NOLINI
05478
                 template<> struct ConwayPolynomial<577, 7> { using ZPZ = aerobus::zpz<577>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<572*; }; // NOLINT template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type =
05479
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<32; //
05480
                 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<17>, ZPZV<576>, ZPZV<449>, ZPZV<572»;
          }; // NOLINT
05481
                 template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
          POLYV<ZPZV<1>, ZPZV<585»; }; // NOLINT
05482
                  template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
          POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<587, 3> { using ZPZ = aerobus::zpz<587>; using type =
05483
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585»; }; // NOLINT
template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<444>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<587, 5> { using ZPZ = aerobus::zpz<587>; using type =
05484
05485
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<585»; }; // NOLINT
                 template<> struct ConwayPolynomial<587, 6> { using ZPZ = aerobus::zpz<587>; using type =
05486
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<204>, ZPZV<121>, ZPZV<226>, ZPZV<2»; }; // NOLINT
05487
                 template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type =
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<585»; }; // NOLINT template<> struct ConwayPolynomial<587, 8> { using ZPZ = aerobus::zpz<587>; using type =
05488
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<492>, ZPZV<444>, ZPZV<91>, ZPZV<2»; };
          NOLINT
05489
                 template<> struct ConwayPolynomial<587, 9> { using ZPZ = aerobus::zpz<587>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<585»;
          }; // NOLINT
05490
                  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 =
          POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3»; }; // NOLINT
05492
                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 =
05493
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3»; };
            template<> struct ConwayPolynomial<593, 5> { using ZPZ = aerobus::zpz<593>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<590»; }; // NOLINT
            template<> struct ConwayPolynomial<593, 6> { using ZPZ = aerobus::zpz<593>; using type =
05495
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<345>, ZPZV<65>, ZPZV<478>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<593, 7> { using ZPZ = aerobus::zpz<593>; using type
05496
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<590»; }; // NOLINT
05497
            template<> struct ConwayPolynomial<593, 8> { using ZPZ = aerobus::zpz<593>;
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<350>, ZPZV<291>, ZPZV<495>, ZPZV<495), }; //
05498
           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<5, ZPZV<8>, ZPZV<223>, ZPZV<523>, ZPZV<590»;
       }; // NOLINT
  template<> struct ConwayPolynomial<599, 1> { using ZPZ = aerobus::zpz<599>; using type =
       POLYV<ZPZV<1>, ZPZV<592»; }; // NOLINT
05500
           template<> struct ConwayPolynomial<599, 2> { using ZPZ = aerobus::zpz<599>; using type =
       POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
05501
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<592»; }; // NOLINT
            template<> struct ConwayPolynomial<599, 4> { using ZPZ = aerobus::zpz<599>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7»; }; // NOLINT
05503
            template<> struct ConwayPolynomial<599, 5> { using ZPZ = aerobus::zpz<599>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<592»; }; // NOLINT template<> struct ConwayPolynomial<599, 6> { using ZPZ = aerobus::zpz<599>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7»; }; // NOLINT
05504
            template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592»; };
           template<> struct ConwayPolynomial<599, 8> { using ZPZ = aerobus::zpz<599>; using type =
05506
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<440>, ZPZV<37>, ZPZV<124>, ZPZV<7»; };
       NOLINT
           template<> struct ConwayPolynomial<599, 9> { using ZPZ = aerobus::zpz<599>; using type =
05507
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<114>, ZPZV<98>, ZPZV<592»;
       }; // NOLINT
            template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type =
05508
       POLYV<ZPZV<1>, ZPZV<594»; }; // NOLINT
            template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
05509
       POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<601, 3> { using ZPZ = aerobus::zpz<601>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<594»; }; // NOLINT template<> struct ConwayPolynomial<601, 4> { using ZPZ = aerobus::zpz<601>; using type =
05511
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<601, 5> { using ZPZ = aerobus::zpz<601>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<594»; }; // NOLINT
05512
05513
            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
05514
            template<> struct ConwayPolynomial<601, 7> { using ZPZ = aerobus::zpz<601>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>4, ZPZV<594*; }; // NOLINT template<> struct ConwayPolynomial<601, 8> { using ZPZ = aerobus::zpz<601>; using type =
05515
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<241>, ZPZV<490>, ZPZV<7»; }; //
       NOLINT
05516
            template<> struct ConwayPolynomial<601, 9> { using ZPZ = aerobus::zpz<601>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<487>, ZPZV<590>, ZPZV<594»;
       }; // NOLINT
05517
            template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
       POLYV<ZPZV<1>, ZPZV<604»; }; // NOLINT
            template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
05518
       POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
05519
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<604»; }; // NOLINT template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
05520
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<607, 5> { using ZPZ = aerobus::zpz<607>; using type =
05521
       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 =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<45>, ZPZV<478>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type =
05523
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<60, ZPZV<604»; }; // NOLINT template<> struct ConwayPolynomial<607, 8> { using ZPZ = aerobus::zpz<607>; using type =
05524
       POLYV<2PZV<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 =
05525
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<444>, ZPZV<429>, ZPZV<604»;
       }; // NOLINT
05526
            template<> struct ConwayPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
       POLYV<ZPZV<1>, ZPZV<611»; }; // NOLINT
            template<> struct ConwayPolynomial<613, 2> { using ZPZ = aerobus::zpz<613>; using type =
       POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; }; // NOLINT
05528
            template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<611»; }; // NOLINT template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
05529
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2; }; // NOLINT template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<611»; }; // NOLINT
05531
           template<> struct ConwayPolynomial<613, 6> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<609>, ZPZV<595>, ZPZV<601>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<61»; }; // NOLINT
05532
```

```
template<> struct ConwayPolynomial<613, 8> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<489>, ZPZV<57>, ZPZV<539>, ZPZV<2»; }; //
         NOLINT
        template<> struct ConwayPolynomial<613, 9> { using ZPZ = aerobus::zpz<613>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<513>, ZPZV<516>, ZPZV<611»;</pre>
05534
         }; // NOLINT
               template<> struct ConwayPolynomial<617, 1> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<614»; }; // NOLINT
              template<> struct ConwayPolynomial<617, 2> { using ZPZ = aerobus::zpz<617>; using type =
05536
        POLYY<ZPZV<1>, ZPZV<612>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
05537
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<614»; }; // NOLINT
              template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type =
05538
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3»; }; // NOLINT
05539
              template<> struct ConwayPolynomial<617, 5> { using ZPZ = aerobus::zpz<617>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<614»; }; // NOLINT template<> struct ConwayPolynomial<617, 6> { using ZPZ = aerobus::zpz<617>; using type =
05540
        POLYYCZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<318>, ZPZV<355>, ZPZV<310>, ZPZV<318>, ZPZV<310>, 
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<614»; };
              template<> struct ConwayPolynomial<617, 8> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<501>, ZPZV<155>, ZPZV<155>, ZPZV<3»; }; //
         NOLINT
        template<> struct ConwayPolynomial<617, 9> { using ZPZ = aerobus::zpz<617>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<515>, ZPZV<388>, ZPZV<543>, ZPZV<614»;</pre>
05543
         }; // NOLINT
              template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
05544
        POLYV<ZPZV<1>, ZPZV<617»; }; // NOLINT
05545
               template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
        POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
05546
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617»; }; // NOLINT
               template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
05547
        05548
               template<> struct ConwayPolynomial<619, 5> { using ZPZ = aerobus::zpz<619>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<617»; }; // NOLINT
        template<> struct ConwayPolynomial<619, 6> { using ZPZ = aerobus::zpz<619>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<468>, ZPZV<347>, ZPZV<2»; }; // NOLINT
05549
               template<> struct ConwayPolynomial<619, 7> { using ZPZ = aerobus::zpz<619>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<617»; };
05551
              template<> struct ConwayPolynomial<619, 8> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<416>, ZPZV<383>, ZPZV<225>, ZPZV<2»; }; //
         NOLINT
05552
              template<> struct ConwayPolynomial<619, 9> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<310>, ZPZV<617»;
         }; // NOLINT
05553
              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 =
05554
         POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3»; }; // NOLINT
05555
               template<> struct ConwayPolynomial<631, 3> { using ZPZ = aerobus::zpz<631>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
05556
               template<> struct ConwayPolynomial<631, 4> { using ZPZ = aerobus::zpz<631>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<3%; ); // NOLINT template<> struct ConwayPolynomial<631, 5> { using ZPZ = aerobus::zpz<631>; using type =
05557
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<52, ZPZV<628»; }; // NOLINT template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<516>, ZPZV<541>, ZPZV<106>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<631, 7> { using ZPZ = aerobus::zpz<631>; using type =
05559
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<628»; }; // NOLIN template<> struct ConwayPolynomial<631, 8> { using ZPZ = aerobus::zpz<631>; using type =
05560
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<3»; }; //
         NOLINT
              template<> struct ConwayPolynomial<631, 9> { using ZPZ = aerobus::zpz<631>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<296>, ZPZV<413>, ZPZV<628»;
         }; // NOLINT
05562
               template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
        POLYV<ZPZV<1>, ZPZV<638»; }; // NOLINT
              template<> struct ConwayPolynomial<641, 2> { using ZPZ = aerobus::zpz<641>; using type =
05563
        POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
05564
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<638»; }; // NOLINT template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type =
05565
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<641, 5> { using ZPZ = aerobus::zpz<641>; using type =
05566
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<638»; }; // NOLINT
              template<> struct ConwayPolynomial<641, 6> { using ZPZ = aerobus::zpz<641>; using type =
05567
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<557>, ZPZV<294>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type =
05568
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<638»; }; // NOLINT
              template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>; using type
05569
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<33»; }; //
05570
              template<> struct ConwayPolynomial<641, 9> { using ZPZ = aerobus::zpz<641>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<66>, ZPZV<141>, ZPZV<638»;
        }; // NOLINT
05571
              template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<632»; };
               template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
        POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11»; }; // NOLINT
              template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
05573
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<632»; }; // NOLINT template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
05574
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<600>, ZPZV<11»; }; // NOLINT
05575
               template<> struct ConwayPolynomial<643, 5> { using ZPZ = aerobus::zpz<643>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<632»; }; // NOLINT
05576
              template<> struct ConwayPolynomial<643, 6> { using ZPZ = aerobus::zpz<643>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<345>, ZPZV<412>, ZPZV<293>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<643, 7> { using ZPZ = aerobus::zpz<643>; using type
05577
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<632»; };
               template<> struct ConwayPolynomial<643, 8> { using ZPZ = aerobus::zpz<643>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<57>, ZPZV<573>, ZPZV<569>, ZPZV<11»; }; //
              template<> struct ConwayPolynomial<643, 9> { using ZPZ = aerobus::zpz<643>; using type =
05579
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<591>, ZPZV<4755, ZPZV<632»;
         }; // NOLINT
               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 =
05581
        POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<647, 3> { using ZPZ = aerobus::zpz<647>; using type =
05582
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<642»; }; // NOLINT template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5»; }; // NOLINT
05584
              template<> struct ConwayPolynomial<647, 5> { using ZPZ = aerobus::zpz<647>; using type =
        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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<308>, ZPZV<385>, ZPZV<642>, ZPZV<5»; }; // NOLINT
05585
05586
               template<> struct ConwayPolynomial<647,
                                                                         7> { using ZPZ = aerobus::zpz<647>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<642»; }; //
05587
             template<> struct ConwayPolynomial<647, 8> { using ZPZ = aerobus::zpz<647>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<603>, ZPZV<259>, ZPZV<271>, ZPZV<5»; }; //
         NOLINT
        template<> struct ConwayPolynomial<647, 9> { using ZPZ = aerobus::zpz<647>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<561>, ZPZV<123>, ZPZV<642»;
05588
         }; // NOLINT
               template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
05589
        POLYV<ZPZV<1>, ZPZV<651»; }; // NOLINT
              template<> struct ConwayPolynomial<653, 2> { using ZPZ = aerobus::zpz<653>; using type =
05590
        POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2»: }: // NOLINT
              template<> struct ConwayPolynomial<653, 3> { using ZPZ = aerobus::zpz<653>; using type =
05591
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<651»; }; // NOLINT template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type =
05592
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
05593
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651»; }; // NOLINT
              template<> struct ConwayPolynomial<653, 6> { using ZPZ = aerobus::zpz<653>; using type =
05594
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<45>, ZPZV<220>, ZPZV<242>, ZPZV<24>, ZPZV<2*; }; // NOLINT
               template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<651»; };
05596
              template<> struct ConwayPolynomial<653, 8> { using ZPZ = aerobus::zpz<653>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<38>, 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
         }; // NOLINT
05598
               template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<657»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
05599
        POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<657»; }; // NOLINT template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
05601
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
05602
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<657»; }; // NOLINT
05603
               template<> struct ConwayPolynomial<659, 6> { using ZPZ = aerobus::zpz<659>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<371>, ZPZV<105>, ZPZV<223>, ZPZV<2»; }; // NOLINT
05604
             template<> struct ConwayPolynomial<659, 7> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<657»; }; // NOLINT template<> struct ConwayPolynomial<659, 8> { using ZPZ = aerobus::zpz<659>; using type =
05605
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<358>, ZPZV<246>, ZPZV<90>, ZPZV<2»; }; //
              template<> struct ConwayPolynomial<659, 9> { using ZPZ = aerobus::zpz<659>; using type =
05606
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592>, ZPZV<46>, ZPZV<657»;
         }; // NOLINT
05607
              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 =
        POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2»; }; // NOLINT
05609
              template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<659»; }; // NOLINT template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2»; }; // NOLINT
05610
```

```
05611
                      template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<659»; }; // NOLINT
                     template<> struct ConwayPolynomial<661, 6> { using ZPZ = aerobus::zpz<661>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<456>, ZPZV<382>, ZPZV<2x; }; // NOLINT template<> struct ConwayPolynomial<661, 7> { using ZPZ = aerobus::zpz<661>; using type
05613
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<659»; };
                                                                                                                                                                                                     // NOLINT
                      template<> struct ConwayPolynomial<661, 8> { using ZPZ = aerobus::zpz<661; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<612>, ZPZV<285>, ZPZV<72>, ZPZV<2»; };
             template<> struct ConwayPolynomial<661, 9> { using ZPZ = aerobus::zpz<661>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<389>, ZPZV<389>, ZPZV<220>, ZPZV<659»;
05615
             }; // NOLINT
05616
                      template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
             POLYV<ZPZV<1>, ZPZV<668»; }; // NOLINT
05617
                      template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
             POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
05618
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<668»; }; // NOLINT
                      template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5»; }; // NOLINT
                      template<> struct ConwayPolynomial<673, 5> { using ZPZ = aerobus::zpz<673>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<668»; }; // NOLINT
                     template<> struct ConwayPolynomial<673, 6> { using ZPZ = aerobus::zpz<673>; using type =
05621
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<524>, ZPZV<248>, ZPZV<35>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<673, 7> { using ZPZ = aerobus::zpz<673>; using type
05622
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                     template<> struct ConwayPolynomial<673, 8> { using ZPZ = aerobus::zpz<673>; using type =
05623
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<669>, ZPZV<587>, ZPZV<302>, ZPZV<5»; };
             NOLINT
05624
                     template<> struct ConwayPolynomial<673, 9> { using ZPZ = aerobus::zpz<673>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<347>, ZPZV<553>, ZPZV<668»;
             }; // NOLINT
05625
                      template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
             POLYV<ZPZV<1>, ZPZV<675»; }; // NOLINT
05626
                      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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<675»; }; // NOLINT
05627
05628
                      template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>; ZPZV<6, 
05629
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<675»; }; // NOLINT template<> struct ConwayPolynomial<677, 6> { using ZPZ = aerobus::zpz<677>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446>, ZPZV<632>, ZPZV<50>, ZPZV<2»; }; // NOLINT
05630
                      template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<675»; };
05632
                    template<> struct ConwayPolynomial<677, 8> { using ZPZ = aerobus::zpz<677>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<363>, ZPZV<619>, ZPZV<152>, ZPZV<2»; }; //
             NOLINT
                      template<> struct ConwayPolynomial<677, 9> { using ZPZ = aerobus::zpz<677>; using type =
05633
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<504>, ZPZV<404>, ZPZV<675»;
             }; // NOLINT
05634
                      template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
             POLYV<ZPZV<1>, ZPZV<678»; }; // NOLINT
                     template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
05635
             POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; }; // NOLINT
                      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 =
05637
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5»; }; // NOLINT
                     template<> struct ConwayPolynomial<683, 5> { using ZPZ = aerobus::zpz<683>; using type =
05638
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<678»; }; // NOLINT
05639
                      template<> struct ConwayPolynomial<683, 6> { using ZPZ = aerobus::zpz<683>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<644>, ZPZV<109>, ZPZV<434>, ZPZV<5»; }; // NOLINT
05640
                    template<> struct ConwayPolynomial<683, 7> { using ZPZ = aerobus::zpz<683>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<678»; }; // NOLINT template<> struct ConwayPolynomial<683, 8> { using ZPZ = aerobus::zpz<683>; using type =
05641
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<383>, ZPZV<184>, ZPZV<65>, ZPZV<5»; }; //
             NOLINT
05642
                      template<> struct ConwayPolynomial<683, 9> { using ZPZ = aerobus::zpz<683>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<678»;
             }; // NOLINT
05643
                      template<> struct ConwayPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
             POLYV<ZPZV<1>, ZPZV<688»; }; // NOLINT
                      template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
05644
             POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; }; // NOLINT
                      template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
05645
            POLYV<2PZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<688»; }; // NOLINT

template<> struct ConwayPolynomial<691, 4> { using ZPZ = aerobus::zpz<691>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; }; // NOLINT

template<> struct ConwayPolynomial<691, 5> { using ZPZ = aerobus::zpz<691>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>; ZPZV<3»; }; // NOLINT
05646
05647
             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 =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<408>, ZPZV<262>, ZPZV<3»; }; // NOLINT
05649
                    template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05650
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<356>, ZPZV<425>, ZPZV<321>, ZPZV<3*; }; //
05651
              template<> struct ConwayPolynomial<691, 9> { using ZPZ = aerobus::zpz<691>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<55>, ZPZV<556>, ZPZV<443>, ZPZV<688»;
         }; // NOLINT
05652
               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 =
05653
        POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; }; // NOLINT
05654
              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 =
05655
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699»; }; // NOLINT
05657
               template<> struct ConwayPolynomial<701, 6> { using ZPZ = aerobus::zpz<701>; using type =
        POLYV<ZPZV<1>, ZPZV<0, ZPZV<1>, ZPZV<571>, ZPZV<327>, ZPZV<285>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<701, 7> { using ZPZ = aerobus::zpz<701>; using type
05658
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<699»; }; // NOLINT
              template<> struct ConwayPolynomial<701, 8> { using ZPZ = aerobus::zpz<701>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<619>, ZPZV<206>, ZPZV<593>, ZPZV<593>, ZPZV<2»; }; //
         NOLINT
              template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type =
0.5660
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<45>, ZPZV<459>, ZPZV<373>, ZPZV<699»;
         }; // NOLINT
05661
               template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
         POLYV<ZPZV<1>, ZPZV<707»; }; // NOLINT
               template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
05662
        POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
05663
        POLYVCZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<707»; }; // NOLINT template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
05664
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<709, 5> { using ZPZ = aerobus::zpz<709>; using type =
05665
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<707»; }; // NOLINT template<> struct ConwayPolynomial<709, 6> { using ZPZ = aerobus::zpz<709>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<669>, ZPZV<514>, ZPZV<295>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<709, 7> { using ZPZ = aerobus::zpz<709>; using type =
05666
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<707»; };
              template<> struct ConwayPolynomial<709, 8> { using ZPZ = aerobus::zpz<709>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<689>, ZPZV<233>, ZPZV<79>, ZPZV<2»; }; //
        NOLINT
              template<> struct ConwayPolynomial<709, 9> { using ZPZ = aerobus::zpz<709>; using type =
05669
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25, ZPZV<257>, ZPZV<257>, ZPZV<2171>, ZPZV<707»;
05670
              template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
        POLYV<ZPZV<1>, ZPZV<708»; }; // NOLINT
              template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
05671
        POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; }; // NOLINT
        template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
05672
05673
         \verb"POLYV<ZPZV<1>, \ \verb"ZPZV<0>, \ \verb"ZPZV<5>, \ \verb"ZPZV<602>, \ \verb"ZPZV<11"; \ \verb"}; \ \ // \ \verb"NOLINT" 
05674
              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
               template<> struct ConwayPolynomial<719, 6> { using ZPZ = aerobus::zpz<719>; using type =
05675
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<533>, ZPZV<591>, ZPZV<182>, ZPZV<11»; }; // NOLINT
              template<> struct ConwayPolynomial<719, 7> { using ZPZ = aerobus::zpz<719>; using type
05676
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<1>, ZPZY<708»; }; // NOLINT template<> struct ConwayPolynomial<719, 8> { using ZPZ = aerobus::zpz<719>; using type =
05677
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<714>, ZPZV<362>, ZPZV<244>, ZPZV<11»; }; //
         NOLINT
05678
               template<> struct ConwayPolynomial<719, 9> { using ZPZ = aerobus::zpz<719>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<288>, ZPZV<260, ZPZV<708»;
         }; // NOLINT
05679
              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 =
05680
        POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial</ri>
727, 3> { using ZPZ = aerobus::zpz<727>; using type =
05681
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<722»; }; // NOLINT template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
05682
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<723>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<727, 5> { using ZPZ = aerobus::zpz<727>; using type =
05683
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, 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
05685
              template<> struct ConwayPolynomial<727, 7> { using ZPZ = aerobus::zpz<727>; using type =
        template<> struct ConwayFolynomia1
FOLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>; j; // NOLINT
template<> struct ConwayPolynomia1
for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFolynomia1

for a struct SonwayFol
05686
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<639>, ZPZV<671>, ZPZV<368>, ZPZV<5»; }; //
        template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<573>, ZPZV<573>, ZPZV<502>, ZPZV<722»;
         }; // NOLINT
               template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
05688
         POLYV<ZPZV<1>, ZPZV<727»; }; // NOLINT
```

```
05689
               template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
        POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; }; // NOLINT
05690
              template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727»; }; // NOLINT template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
05691
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<5; }; // NOLINT 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
              template<> struct ConwayPolynomial<733, 6> { using ZPZ = aerobus::zpz<733>; using type =
05693
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1A, ZPZV<549>, ZPZV<5151>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<733, 7> { using ZPZ = aerobus::zpz<733>; using type =
05694
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<727»; }; // NOLINT
              template<> struct ConwayPolynomial<733, 8> { using ZPZ = aerobus::zpz<733>; using type =
05695
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<532>, ZPZV<610>, ZPZV<142>, ZPZV<6»; }; //
         NOLINT
05696
              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
05697
               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 =
        POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
05699
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; }; // NOLINT template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
05700
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<3»; }; // NOLINT
05701
              template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<736»; }; // NOLINT
05702
               template<> struct ConwayPolynomial<739, 6> { using ZPZ = aerobus::zpz<739>; using type =
        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
05703
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<44>, ZPZV<736»; };
              template<> struct ConwayPolynomial<739, 8> { using ZPZ = aerobus::zpz<739>; using type
05704
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<401>, ZPZV<169>, ZPZV<25>, ZPZV<3»; };
         NOLINT
05705
              template<> struct ConwayPolynomial<739, 9> { using ZPZ = aerobus::zpz<739>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<6166, ZPZV<81>, ZPZV<736»;
         }; // NOLINT
05706
               template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<738»; }; // NOLINT
05707
             template<> struct ConwayPolynomial<743, 2> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<743, 3> { using ZPZ = aerobus::zpz<743>; using type =
05708
        POLYV<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<38*, }; // NOLINT
template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<5»; }; // NOLINT
05710
             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 =
05711
        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<7
                                                                                                                     43>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
05713
              template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>; }; //
         NOLINT
        template<> struct ConwayPolynomial<743, 9> { using ZPZ = aerobus::zpz<743>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<327>, ZPZV<3738»;</pre>
05714
        }; // NOLINT
               template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
05715
        POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
              template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
05716
        POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZV<748»; }; // NOLINT template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
05718
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<525>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<751, 5> { using ZPZ = aerobus::zpz<751>; using type =
05719
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT template<> struct ConwayPolynomial<751, 6> { using ZPZ = aerobus::zpz<751>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<298>, ZPZV<633>, ZPZV<539>, ZPZV<539>, ZPZV<539>,
              template<> struct ConwayPolynomial<751, 7> { using ZPZ = aerobus::zpz<751>; using type =
05721
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT
05722
              template<> struct ConwayPolynomial<751, 8> { using ZPZ = aerobus::zpz<751>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<741>, ZPZV<243>, 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<0>, ZPZV<10>, ZPZV<703>, ZPZV<489>, ZPZV<748»;
         }; // NOLINT
05724
              template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
        POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
              template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
05725
        POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<755»; };
                                                                              // NOLINT
             template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
05727
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<757, 5> { using ZPZ = aerobus::zpz<757>; using type =
05728
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<755»; };
      template<> struct ConwayPolynomial<757, 6> { using ZPZ = aerobus::zpz<757>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<753>, ZPZV<745>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<757, 7> { using ZPZ = aerobus::zpz<757>; using type =
05730
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<757, 8> { using ZPZ = aerobus::zpz<757>; using type =
05731
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<110>, ZPZV<509>, ZPZV<2»; }; //
05732
           template<> struct ConwayPolynomial<757, 9> { using ZPZ = aerobus::zpz<757>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<688>, ZPZV<702>, ZPZV<705»;
       }; // NOLINT
05733
            template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
       POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
            template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
       POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; }; // NOLINT
05735
           template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<6»; }; // NOLINT
05736
            template<> struct ConwayPolynomial<761, 5> { using ZPZ = aerobus::zpz<761>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT
05738
           template<> struct ConwayPolynomial<761, 6> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<634>, ZPZV<597>, ZPZV<155>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<761, 7> { using ZPZ = aerobus::zpz<761>; using type
05739
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<755»; }; // NOLINT
            template<> struct ConwayPolynomial<761, 8> { using ZPZ = aerobus::zpz<761; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<603>, ZPZV<144>, ZPZV<540>, ZPZV<54»; }; //
       NOLINT
       template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<317>, ZPZV<571>, ZPZV<755»;</pre>
05741
       }; // NOLINT
05742
            template<> struct ConwayPolynomial<769, 1> { using ZPZ = aerobus::zpz<769>; using type =
       POLYV<ZPZV<1>, ZPZV<758»; }; // NOLINT
05743
           template<> struct ConwayPolynomial<769, 2> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
05744
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; }; // NOLINT
            template<> struct ConwayPolynomial</69, 4> { using ZPZ = aerobus::zpz<769>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 5> { using ZPZ = aerobus::zpz<769>; using type =
05746
      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»; }; // NOLINT</pre>
05747
05748
           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<8>, ZPZV<758»; }; // NOLINT
05749
            template<> struct ConwayPolynomial<769, 8> { using ZPZ = aerobus::zpz<769>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<560>, ZPZV<574>, ZPZV<632>, ZPZV<11»; }; //
       NOLINT
           template<> struct ConwayPolynomial<769, 9> { using ZPZ = aerobus::zpz<769>; using type =
05750
       POLYV<ZPZV<1>, 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 =
05751
       POLYV<ZPZV<1>, ZPZV<771»; }; // NOLINT template<> struct ConwayPolynomial<773, 2> { using ZPZ = aerobus::zpz<773>; using type =
05752
      POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial773, 3> { using ZPZ = aerobus::zpz<773>; using type =
05753
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<771»; }; // NOLINT
           template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
05754
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2w; }; // NOLINT template<> struct ConwayPolynomial<773, 5> { using ZPZ = aerobus::zpz<773>; using type =
05755
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<771»; }; // NOLINT template<> struct ConwayPolynomial<773, 6> { using ZPZ = aerobus::zpz<773>; using type =
05756
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<91>, ZPZV<3>, ZPZV<581>, ZPZV<2»; };
                                                                                                 // NOLINT
            template<> struct ConwayPolynomial<773, 7> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<771»; };
05758
           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»; }; //
       NOLINT
           template<> struct ConwayPolynomial<773, 9> { using ZPZ = aerobus::zpz<773>; using type =
05759
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<216>, ZPZV<574>, ZPZV<771»;
       }; // NOLINT
05760
           template<> struct ConwayPolynomial<787, 1> { using ZPZ = aerobus::zpz<787>; using type =
       POLYV<ZPZV<1>, ZPZV<785»; }; // NOLINT
            template<> struct ConwayPolynomial<787, 2> { using ZPZ = aerobus::zpz<787>; using type =
05761
       POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
05762
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785»; }; // NOLINT template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type =
05763
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
05764
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<785»; }; // NOLINT
            template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<98>, ZPZV<512>, ZPZV<606>, ZPZV<2»; }; // NOLINI
           template<> struct ConwayPolynomial<787, 7> { using ZPZ = aerobus::zpz<787>; using type =
05766
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, 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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<612>, ZPZV<26>, ZPZV<715>, ZPZV<2»; }; //
05767
```

```
NOLINT
         template<> struct ConwayPolynomial<787, 9> { using ZPZ = aerobus::zpz<787>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<480>, ZPZV<573>, ZPZV<785»;
         }; // NOLINT
05769
                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 =
         POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»; }; // NOLINT
05771
               template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type =
05772
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<717>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<797, 5> { using ZPZ = aerobus::zpz<797>; using type =
05773
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<795»; }; // NOLINT
05774
              template<> struct ConwayPolynomial<797, 6> { using ZPZ = aerobus::zpz<797>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<396>, ZPZV<71>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<797, 7> { using ZPZ = aerobus::zpz<797>; using type =
05775
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<795»; }; // NOLINT
               template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2*; }; //
05777
               template<> struct ConwayPolynomial<797, 9> { using ZPZ = aerobus::zpz<797>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<240>, ZPZV<599>, ZPZV<795»;
         }; // NOLINT
05778
                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 =
05779
         POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
05780
               template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<806»; }; // NOLINT
template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; }; // NOLINT
05781
               template<> struct ConwayPolynomial<809, 5> { using ZPZ = aerobus::zpz<809>; using type =
05782
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; }; // NOLINT
05783
               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 template<> struct ConwayPolynomial<809, 7> { using ZPZ = aerobus::zpz<809>; using type =
05784
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; }; // NOLINT
               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>; //
         NOLINT
         template<> struct ConwayPolynomial<809, 9> { using ZPZ = aerobus::zpz<809>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3+>, ZPZV<341>, ZPZV<341>, ZPZV<727>, ZPZV<806»;</pre>
05786
         }; // NOLINT
    template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
         POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
05788
              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 =
05789
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<808»; }; // NOLINT template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type =
05790
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3»; }; // NOLINT
05791
               template<> struct ConwayPolynomial<811, 5> { using ZPZ = aerobus::zpz<811>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<808»; }; // NOLINT template<> struct ConwayPolynomial<811, 6> { using ZPZ = aerobus::zpz<811>; using type =
05792
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<780>, ZPZV<755>, ZPZV<307>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<811, 7> { using ZPZ = aerobus::zpz<811>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<808»; };
              template<> struct ConwayPolynomial<811, 8> { using ZPZ = aerobus::zpz<811>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<3»; }; //
         NOLINT
         template<> struct ConwayPolynomial<811, 9> { using ZPZ = aerobus::zpz<811>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<382>, ZPZV<200>, ZPZV<808*;
05795
         }; // NOLINT
              template<> struct ConwayPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
05796
         POLYV<ZPZV<1>, ZPZV<819»; }; // NOLINT
05797
               template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
         POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<821, 3> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<219»; }; // NOLINT template<> struct ConwayPolynomial<821, 4> { using ZPZ = aerobus::zpz<821>; using type =
05799
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<662>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
0.5800
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<819»; }; // NOLINT
         template<> struct ConwayPolynomial<821, 6> { using ZPZ = aerobus::zpz<821>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<160>, ZPZV<130>, ZPZV<803>, ZPZV<2»; }; // NOLINT
05801
               template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type
05802
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<819»; }; // NOLINT template<> struct ConwayPolynomial<821, 8> { using ZPZ = aerobus::zpz<821>; using type =
05803
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<55>, ZPZV<556>, ZPZV<589>, ZPZV<589>, ZPZV<2»; }; //
         NOLINT
05804
               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<0>, ZPZV<5>, ZPZV<650>, ZPZV<650>, ZPZV<557>, ZPZV<819»;
         }; // NOLINT
05805
               \texttt{template<> struct ConwayPolynomial<823, 1> \{ using \ ZPZ = aerobus::zpz<823>; using \ type = aerobus::zp
         POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
               template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; };
            template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
05808
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
05809
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
05810
           template<> struct ConwayPolynomial<823, 6> { using ZPZ = aerobus::zpz<823>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<82>, ZPZV<616>, ZPZV<744>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type =
05811
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2D, ZPZV<2D, ZPZV<2D, ZPZV<820»; }; // NOLINT template<> struct ConwayPolynomial<823, 8> { using ZPZ = aerobus::zpz<823>; using type =
05812
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<451>, ZPZV<437>, ZPZV<31>, ZPZV<3»; };
05813
           template<> struct ConwayPolynomial<823, 9> { using ZPZ = aerobus::zpz<823>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<740>, ZPZV<609>, ZPZV<820»;
       }; // NOLINT
            template<> struct ConwayPolynomial<827, 1> { using ZPZ = aerobus::zpz<827>; using type =
05814
       POLYV<ZPZV<1>, ZPZV<825»; }; // NOLINT
           template<> struct ConwayPolynomial<827, 2> { using ZPZ = aerobus::zpz<827>; using type =
       POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT
05816
           template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825»; }; // NOLINT template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2»; }; // NOLINT
05817
            template<> struct ConwayPolynomial<827, 5> { using ZPZ = aerobus::zpz<827>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<825»; }; // NOLINT
05819
           template<> struct ConwayPolynomial<827, 6> { using ZPZ = aerobus::zpz<827>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<685>, ZPZV<601>, ZPZV<691>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<827, 7> { using ZPZ = aerobus::zpz<827>; using type =
05820
      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<827, 8> { using ZPZ = aerobus::zpz<827>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<812>, ZPZV<79>, ZPZV<32>, ZPZV<32>; };
       template<> struct ConwayPolynomial<827, 9> { using ZPZ = aerobus::zpz<827>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<372>, ZPZV<825»;</pre>
05822
       }; // NOLINT
            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 =
05825
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<827»; }; // NOLINT template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
05826
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<604>, ZPZV<2»; }; // NOLINT
05827
           template<> struct ConwayPolynomial<829, 5> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<827»; }; // NOLINT
05828
           template<> struct ConwayPolynomial<829, 6> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<476>, ZPZV<817>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type =
05829
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<827»; };
           template<> struct ConwayPolynomial<829, 8> { using ZPZ = aerobus::zpz<829>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<468>, ZPZV<241>, ZPZV<138>, ZPZV<2»; }; //
           template<> struct ConwayPolynomial<829, 9> { using ZPZ = aerobus::zpz<829>; using type =
05831
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<621>, ZPZV<552>, ZPZV<827»;
       }; // NOLINT
    template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
05832
       POLYV<ZPZV<1>, ZPZV<828»; }; // NOLINT
           template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
05833
       POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
05834
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; }; // NOLINT
           template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11»; }; // NOLINT
05836
           template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
       template<> struct ConwayPolynomialx839, 6> { using ZPZ = aerobus::zpz<839>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<370>, ZPZV<537>, ZPZV<23>, ZPZV<11»; }; // NOLINT
05837
            template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<828%; }; // NOLIN template<> struct ConwayPolynomial<839, 8> { using ZPZ = aerobus::zpz<839>; using type =
05839
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<553>, ZPZV<779>, ZPZV<329>, ZPZV<11»; }; //
       NOLINT
           template<> struct ConwayPolynomial<839, 9> { using ZPZ = aerobus::zpz<839>; using type =
05840
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<349>, ZPZV<349>, ZPZV<206>, ZPZV<828»;
       }; // NOLINT
05841
           template<> struct ConwayPolynomial<853, 1> { using ZPZ = aerobus::zpz<853>; using type =
       POLYV<ZPZV<1>, ZPZV<851»; }; // NOLINT
           template<> struct ConwayPolynomial<853, 2> { using ZPZ = aerobus::zpz<853>; using type =
05842
       POLYV<ZPZV<1>, ZPZV<852>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<853, 3> { using ZPZ = aerobus::zpz<853>; using type =
      POLYV<ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
05844
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
05845
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<851»; }; // NOLINT
```

```
05846
                    template<> struct ConwayPolynomial<853, 6> { using ZPZ = aerobus::zpz<853>; using type =
            POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<276>, ZPZV<216>, ZPZV<512>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 8> { using ZPZ = aerobus::zpz<853>; using type =
05848
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<846>, ZPZV<846>, ZPZV<118>, ZPZV<2»; }; //
            NOLINT
                    template<> struct ConwayPolynomial<853, 9> { using ZPZ = aerobus::zpz<853>; using type
05849
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<677>, ZPZV<821>, ZPZV<851»;
            }; // NOLINT
05850
                    template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
            POLYV<ZPZV<1>, ZPZV<854»; }; // NOLINT
                    template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
05851
            POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; }; // NOLINT
05852
                   template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; ); // NOLINT
template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<528>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<857, 5> { using ZPZ = aerobus::zpz<857>; using type =
05853
05854
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<854»; }; // NOLINT
                    template<> struct ConwayPolynomial<857, 6> { using ZPZ = aerobus::zpz<857>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<824>, ZPZV<65>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::zpz<857>; using type =
05856
           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
05857
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<611>, ZPZV<552>, ZPZV<494>, ZPZV<3»; }; //
            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<30>, ZPZV<308>, ZPZV<719>, ZPZV<854»;
05858
            }; // NOLINT
                    template<> struct ConwayPolynomial<859, 1> { using ZPZ = aerobus::zpz<859>; using type =
05859
            POLYV<ZPZV<1>, ZPZV<857»; }; // NOLINT
                    template<> struct ConwayPolynomial<859, 2> { using ZPZ = aerobus::zpz<859>; using type =
            POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT
05861
                    template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2»; }; // NOLINT
05862
                    template<> struct ConwayPolynomial<859, 5> { using ZPZ = aerobus::zpz<859>; using type =
05863
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<857»; }; // NOLINT
05864
                   template<> struct ConwayPolynomial<859, 6> { using ZPZ = aerobus::zpz<859>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<419>, ZPZV<646>, ZPZV<566>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpz<859>; using type = DOLYVZPZVZ<1>, ZPZV<20, 
05865
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<857»; };
                    template<> struct ConwayPolynomial<859, 8> { using ZPZ = aerobus::zpz<859>; using type
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<522>, ZPZV<446>, ZPZV<672>, ZPZV<672>, ZPZV<2»; }; //
            template<> struct ConwayPolynomial<859, 9> { using ZPZ = aerobus::zpz<859>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<648>, ZPZV<845>, ZPZV<857»;</pre>
05867
            }; // NOLINT
05868
                    template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
            POLYV<ZPZV<1>, ZPZV<858»; }; // NOLINT
05869
                    template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
            POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
05870
            POLYY<ZPZY<1>, ZPZV<0>, ZPZV<5>, ZPZV<588%; }; // 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 =
05872
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<858»; }; // NOLINT
05873
                   template<> struct ConwayPolynomial<863, 6> { using ZPZ = aerobus::zpz<863>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<330>, ZPZV<300>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<863, 7> { using ZPZ = aerobus::zpz<863>; using type
05874
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<858»; }; // NOLINT
05875
                   template<> struct ConwayPolynomial<863, 8> { using ZPZ = aerobus::zpz<863>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<765>, ZPZV<576>, ZPZV<849>, ZPZV<5*; }; //
            NOLINT
05876
                   template<> struct ConwayPolynomial<863, 9> { using ZPZ = aerobus::zpz<863>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<381>, ZPZV<381>, ZPZV<4858»; };
            // NOLINT
05877
                    template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
            POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
                    template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
05878
            POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomials877, 3> { using ZPZ = aerobus::zpz<877>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875»; }; // NOLINT
05879
                    template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type =
05880
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<877, 5> { using ZPZ = aerobus::zpz<877>; using type =
05881
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<875»; }; // NOLINT template<> struct ConwayPolynomial<877, 6> { using ZPZ = aerobus::zpz<877>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<400>, ZPZV<855>, ZPZV<2»; }; // NOLINT
05882
                    template<> struct ConwayPolynomial<877, 7> { using ZPZ = aerobus::zpz<877>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<875»; };
05884
                   template<> struct ConwayPolynomial<877, 8> { using ZPZ = aerobus::zpz<877>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<767>, ZPZV<319>, ZPZV<347>, ZPZV<2w; }; //
            NOLTNT
```

```
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»;
         }; // NOLINT
05886
               template<> struct ConwayPolynomial<881, 1> { using ZPZ = aerobus::zpz<881>; using type =
         POLYV<ZPZV<1>, ZPZV<878»; }; // NOLINT
               template<> struct ConwayPolynomial<881, 2> { using ZPZ = aerobus::zpz<881>; using type =
05887
         POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
05888
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<67>, j; // NOLINT
template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<447>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<881, 5> { using ZPZ = aerobus::zpz<881>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3»; }; // NOLINT
Template<> struct ConwayPolynomial<881, 5> { using ZPZ = aerobus::zpz<881>; using type =
05889
05890
         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 =
05891
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<419>, ZPZV<231>, ZPZV<3»; }; // NOLINT
05892
               template<> struct ConwayPolynomial<881, 7> { using ZPZ = aerobus::zpz<881>; using type =
         Completes Struct Conmaylolynomial vol, // ( using ZPZ - defodus::ZpZvool/, using Lype = POLYV-ZPZV-1>, ZPZV-0>, ZPZV-0>,
05893
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<635>, ZPZV<490>, ZPZV<561>, ZPZV<561>, ZPZV<3»; }; //
         template<> struct ConwayPolynomial<881, 9> { using ZPZ = aerobus::zpz<881>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<587>, ZPZV<510>, ZPZV<878»;
05894
         }; // NOLINT
               template<> struct ConwayPolynomial<883, 1> { using ZPZ = aerobus::zpz<883>; using type =
05895
         POLYV<ZPZV<1>, ZPZV<881»; }; // NOLINT
               template<> struct ConwayPolynomial<883, 2> { using ZPZ = aerobus::zpz<883>; using type =
         POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; }; // NOLINT
05897
               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 =
05898
        POLYVCZPZV<1>, ZPZV<2>, ZPZV<8>, ZPZV<715>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<883, 5> { using ZPZ = aerobus::zpz<883>; using type =
05899
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<881»; }; // NOLINT
05900
              template<> struct ConwayPolynomial<883, 6> { using ZPZ = aerobus::zpz<883>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<87>, ZPZV<865>, ZPZV<871>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type =
05901
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<8*), ZPZV<6>, ZPZV<8*); // NOLINT template<> struct ConwayPolynomial<883, 8> { using ZPZ = aerobus::zpz<883>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<740>, ZPZV<762>, ZPZV<768>, ZPZV<20»; };
05903
              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<360>, ZPZV<557>, ZPZV<881»;
         }; // NOLINT
05904
               template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
         POLYV<ZPZV<1>, ZPZV<882»; }; // NOLINT
05905
               template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
         POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; }; // NOLINT
05906
              template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<882»; }; // NOLINT
               template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
05907
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<887, 5> { using ZPZ = aerobus::zpz<887>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<882»; }; // NOLINT
05909
               template<> struct ConwayPolynomial<887, 6> { using ZPZ = aerobus::zpz<887>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<775>, ZPZV<474>, ZPZV<28>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<887, 7> { using ZPZ = aerobus::zpz<887>; using type
05910
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<882»; }; //
              template<> struct ConwayPolynomial<887, 8> { using ZPZ = aerobus::zpz<887>; using type =
05911
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<781>, ZPZV<381>, ZPZV<706>, ZPZV<5»; };
         NOLINT
05912
              template<> struct ConwayPolynomial<887, 9> { using ZPZ = aerobus::zpz<887>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<727>, ZPZV<345>, ZPZV<882»;
         }; // NOLINT
05913
                template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
         POLYV<ZPZV<1>, ZPZV<905»; }; // NOLINT
05914
               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 =
05915
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<905»; }; // NOLINT
               template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
05916
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2»; }; // NOLINT
05917
              template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<905; }; // NOLINT template<> struct ConwayPolynomial<907, 6> { using ZPZ = aerobus::zpz<907>; using type =
05918
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<626, ZPZV<266>, ZPZV<266>, ZPZV<266>, ZPZV<29; }; // NOLINT template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; };
05920
              template<> struct ConwayPolynomial<907, 8> { using ZPZ = aerobus::zpz<907>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<584>, ZPZV<518>, ZPZV<811>, ZPZV<81; }; //
         NOLINT
               template<> struct ConwayPolynomial<907, 9> { using ZPZ = aerobus::zpz<907>; using type
05921
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<783>, ZPZV<57>, ZPZV<905»;
         }; // NOLINT
05922
               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 =
05923
         POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; }; // NOLINT
```

```
05924
            template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; }; // NOLINT template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
05925
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
05926
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<29, ZPZV<894»; }; // NOLINT template<> struct ConwayPolynomial<911, 6> { using ZPZ = aerobus::zpz<911>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<172>, ZPZV<683>, ZPZV<19>, ZPZV<17»; }; // NOLINI
           template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
05928
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<894»; }; // NOLINT
05929
           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»; }; //
            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<0>, ZPZV<679>, ZPZV<116>, ZPZV<894»;
       }; // NOLINT
05931
           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 =
       POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912»; }; // NOLINT template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
05934
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<919, 5> { using ZPZ = aerobus::zpz<919>; using type =
05935
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<912»; }; // NOLINT
           template<> struct ConwayPolynomial<919, 6> { using ZPZ = aerobus::zpz<919>; using type =
05936
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<312>, ZPZV<817>, ZPZV<113>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<919, 7> { using ZPZ = aerobus::zpz<919>; using type
05937
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<9>, ZPZV<9\), }; // NOLINT
           template<> struct ConwayPolynomial<919, 8> { using ZPZ = aerobus::zpz<919>; using type =
05938
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<202>, ZPZV<504>, ZPZV<50*, }; //
           template<> struct ConwayPolynomial<919, 9> { using ZPZ = aerobus::zpz<919>; using type =
05939
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<410>, ZPZV<623>, ZPZV<912»;
       }; // NOLINT
       template<> struct ConwayPolynomial<929, 1> { using ZPZ = aerobus::zpz<929>; using type = POLYV<ZPZV<1>, ZPZV<926»; }; // NOLINT
05940
            template<> struct ConwayPolynomial<929, 2> { using ZPZ = aerobus::zpz<929>; using type =
       POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3»; }; // NOLINT
05942
           template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<926»; }; // NOLINT
template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3»; }; // NOLINT
05943
            template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<926»; }; // NOLINT
05945
           template<> struct ConwayPolynomial<929, 6> { using ZPZ = aerobus::zpz<929>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<805>, ZPZV<92>, ZPZV<86>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type =
05946
       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
05947
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699>, ZPZV<292>, ZPZV<586>, ZPZV<3»; }; //
       NOLINT
       template<> struct ConwayPolynomial<929, 9> { using ZPZ = aerobus::zpz<929>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<481>, ZPZV<199>, ZPZV<926»;</pre>
05948
       }; // NOLINT
            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 =
05950
       POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<937, 3> { using ZPZ = aerobus::zpz<937>; using type =
05951
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<932»; }; // NOLINT template<> struct ConwayPolynomial<937, 4> { using ZPZ = aerobus::zpz<937>; using type =
05952
       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<23>, ZPZV<585>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
05953
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<932»; }; // NOLINT
05954
           template<> struct ConwayPolynomial<937, 6> { using ZPZ = aerobus::zpz<937>; using type =
       POLYY<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
05955
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<932»; }; //
           template<> struct ConwayPolynomial<937, 8> { using ZPZ = aerobus::zpz<937>; using type
05956
       POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<658>, 2PZV<26>, 2PZV<53>, 2PZV<5»; };
       NOLINT
           template<> struct ConwayPolynomial<937, 9> { using ZPZ = aerobus::zpz<937>; using type =
05957
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<28>, ZPZV<533>, ZPZV<483>, ZPZV<932»;
       }; // NOLINT
05958
            template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
       POLYV<ZPZV<1>, ZPZV<939»; }; // NOLINT
           template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
05959
       POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
05960
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<939»; }; // NOLINT
            template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2»; }; // NOLINT
05962
           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
05963
           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»; };
                  template<> struct ConwayPolynomial<941, 7> { using ZPZ = aerobus::zpz<941>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<399»; }; // NOLINT
05965
                 template<> struct ConwayPolynomial<941, 8> { using ZPZ = aerobus::zpz<941>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<675>, ZPZV<590>, ZPZV<2»; }; //
          NOLTNT
05966
                  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<0>, ZPZV<1>, ZPZV<708>, ZPZV<708>, ZPZV<197>, ZPZV<939%;
          }; // NOLINT
05967
                  template<> struct ConwayPolynomial<947, 1> { using ZPZ = aerobus::zpz<947>; using type =
          POLYV<ZPZV<1>, ZPZV<945»; }; // NOLINT
                 template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
05968
          POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<945»; }; // NOLINT
         template<> struct ConwayPolynomial<947, 4> { using ZPZ = aerobus::zpz<947>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2»; }; // NOLINT
05970
                  template<> struct ConwayPolynomial<947, 5> { using ZPZ = aerobus::zpz<947>; using type =
05971
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<945»; }; // NOLINT
05972
                  template<> struct ConwayPolynomial<947, 6> { using ZPZ = aerobus::zpz<947>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<880>, ZPZV<787>, ZPZV<95>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<947, 7> { using ZPZ = aerobus::zpz<947>; using type =
05973
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 8> { using ZPZ = aerobus::zpz<947>; using type =
05974
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<845>, ZPZV<597>, ZPZV<581>, ZPZV<2»; }; //
                 template<> struct ConwayPolynomial<947, 9> { using ZPZ = aerobus::zpz<947>; using type =
05975
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<20>, ZPZV<20>,
          }; // NOLINT
05976
                 template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<950»; }; // NOLINT
05977
                  template<> struct ConwayPolynomial<953, 2> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3»; }; // NOLINT
05978
                template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950»; }; // NOLINT template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
05979
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<685>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<953, 5> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<950»; }; // NOLINT
                  template<> struct ConwayPolynomial<953, 6> { using ZPZ = aerobus::zpz<953>; using type =
05981
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<507>, ZPZV<829>, ZPZV<730>, ZPZV<3»; }; // NOLINT
05982
                  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>, ZPZV<5>, ZPZV<950»; }; // NOLINT
                 template<> struct ConwayPolynomial<953, 8> { using ZPZ = aerobus::zpz<953>; using type =
05983
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<579>, ZPZV<658>, ZPZV<108>, ZPZV<3»; }; //
          NOLINT
05984
                 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
05985
                  template<> struct ConwavPolynomial<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 =
          POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5»; }; // NOLINT
                  template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
05987
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
05988
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; };
                                                                                                                // NOLINT
                 template<> struct ConwayPolynomial<967, 5> { using ZPZ = aerobus::zpz<967>; using type =
05989
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<962»; }; // NOLINT
05990
                  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 template<> struct ConwayPolynomial<967, 7> { using ZPZ = aerobus::zpz<967>; using type
05991
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<962»; }; //
                  template<> struct ConwayPolynomial<967, 8> { using ZPZ = aerobus::zpz<967>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<840>, ZPZV<502>, ZPZV<136>, ZPZV<5»; }; //
          NOLINT
05993
          template<> struct ConwayPolynomial<967, 9> { using ZPZ = aerobus::zpz<967>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<512>, ZPZV<512>, ZPZV<783>, ZPZV<962»;</pre>
          }; // NOLINT
05994
                  template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<965»; }; // NOLINT
05995
                 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 =
05996
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<65»; }; // NOLINT template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<527>, ZPZV<6»; }; // NOLINT
05998
                  template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<965»; }; // NOLINT
                  template<> struct ConwayPolynomial<971, 6> { using ZPZ = aerobus::zpz<971>; using type =
05999
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<970>, ZPZV<729>, ZPZV<718>, ZPZV<6»; }; // NOLINT
06000
                  template<> struct ConwayPolynomial<971,
                                                                                         7> { using ZPZ = aerobus::zpz<971>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<0>, ZPZV<1>, ZPZV<95»; }; // NOL template<> struct ConwayPolynomial<971, 8> { using ZPZ = aerobus::zpz<971>; using type = 
06001
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<72>, ZPZV<725>, ZPZV<281>, ZPZV<206>, ZPZV<6»; }; //
          NOLINT
06002
                 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»;
06003
          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 =
06004
      POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; }; // NOLINT
          template<> struct ConwayPolynomial 977, 3> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<974»; }; // NOLINT
          template<> struct ConwayPolynomial<977, 4> { using ZPZ = aerobus::zpz<977>; using type =
06006
      template<> struct ConwayPolynomial<977, 5> { using ZPZ = aerobus::zpz<977>; using type =
06007
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<974»; }; // NOLINT
06008
          template<> struct ConwayPolynomial<977, 6> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<729>, ZPZV<830>, ZPZV<753>, ZPZV<75»; }; // NOLIN
06009
          template<> struct ConwayPolynomial<977, 7> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<74»; }; // NOLIN template<> struct ConwayPolynomial<977, 8> { using ZPZ = aerobus::zpz<977>; using type =
06010
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<85>, ZPZV<807>, ZPZV<77>, ZPZV<3»; };
          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<450>, ZPZV<440>, ZPZV<740>, ZPZV<974»;
      }; // NOLINT
06012
          template<> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<978»; }; // NOLINT
06013
          template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; }; // NOLINT
06014
          template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978»; }; // NOLINT template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
06015
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5s»; }; // NOLINT
template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
06016
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<978»; }; // NOLINT
          template<> struct ConwayPolynomial<983, 6> { using ZPZ = aerobus::zpz<983>; using type =
06017
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<849>, ZPZV<296>, ZPZV<228>, ZPZV<5»; }; // NOLINI
06018
          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»; }; // NOLINT
          template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
06019
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<53»; }; //
      NOLINT
          template<> struct ConwayPolynomial<983, 9> { using ZPZ = aerobus::zpz<983>; using type =
06020
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<85>, ZPZV<858>, ZPZV<87>, ZPZV<978»;
      }; // NOLINT
          template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
06021
      POLYV<ZPZV<1>, ZPZV<985»; }; // NOLINT
          template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; }; // NOLINT
06023
          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 =
06024
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<991, 5> { using ZPZ = aerobus::zpz<991>; using type =
06025
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<985»; }; // NOLINT
06026
          template<> struct ConwayPolynomial<991, 6> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<637>, ZPZV<855>, ZPZV<278>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<991, 7> { using ZPZ = aerobus::zpz<991>; using type =
06027
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<985»; }; // NOLINT
          template<> struct ConwayPolynomial<991, 8> { using ZPZ = aerobus::zpz<991>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<941>, ZPZV<786>, ZPZV<234>, ZPZV<6»; }; //
06029
          template<> struct ConwayPolynomial<991, 9> { using ZPZ = aerobus::zpz<991>; using type :
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<466>, ZPZV<265, ZPZV<985»;
      }; // NOLINT
06030
           template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<990»; }; // NOLINT
06031
          template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; }; // NOLINT
06032
          template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990»; }; // NOLINT
          template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
06033
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7»; }; // NOLINT
          template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
06034
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<990»; }; // NOLINT
06035
          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
06036
      POLYV<ZPZV<1>, 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 =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<934>, ZPZV<473>, ZPZV<241>, ZPZV<241>, ZPZV<7»; }; //
      NOLINT
          template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type =
06038
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<732>, ZPZV<616, ZPZV<990»;
      }; // NOLINT
06039 #endif // DO_NOT_DOCUMENT
06040 }
         // namespace aerobus
06041 #endif // AEROBUS_CONWAY_IMPORTS
06042
06043 #endif // __INC_AEROBUS__ // NOLINT
```

src/examples.h File Reference 9.4

examples.h 9.5

Go to the documentation of this file.

00001 #ifndef SRC_EXAMPLES_H_

00002 #define SRC_EXAMPLES_H_

00050 #endif // SRC_EXAMPLES_H_

Chapter 10

Examples

10.1 examples/hermite.cpp

How to use aerobus::known_polynomials::hermite_phys polynomials

```
#include <cmath>
#include <iostream>
#include "../src/aerobus.h"
namespace standardlib {
    double H3 (double x) {
         return 8 * std::pow(x, 3) - 12 * x;
    double H4(double x) {
         return 16 * std::pow(x, 4) - 48 * x * x + 12;
namespace aerobuslib {
    double H3(double x) {
        return 8 * aerobus::pow_scalar<double, 3>(x) - 12 * x;
    double H4(double x) {
         return 16 * aerobus::pow_scalar<double, 4>(x) - 48 * x * x + 12;
int main() {
    std::cout « std::hermite(3, 10) « '=' « standardlib::H3(10) « '\n' « std::hermite(4, 10) « '=' « standardlib::H4(10) « '\n';
    std::cout « aerobus::known_polynomials::hermite_phys<4>::eval(10) « '=' « aerobuslib::H3(10) « '\n' « aerobus::known_polynomials::hermite_phys<4>::eval(10) « '=' « aerobuslib::H4(10) « '\n';
```

10.2 examples/custom_taylor.cpp

How to implement your own Taylor serie using aerobus::taylor

```
#include <cmath>
#include <iostream>
#include <iomanip>
#include "../src/aerobus.h"

template<typename T, size_t i>
struct my_coeff {
    using type = aerobus::makefraction_t<T, aerobus::bell_t<T, i>, aerobus::factorial_t<T, i>>;

template<size_t deg>
```

206 Examples

```
using F = aerobus::taylor<aerobus::i64, my_coeff, deg>;
int main() {
   constexpr double x = F<15>::eval(0.1);
   double xx = std::exp(std::exp(0.1) - 1);
   std::cout « std::setprecision(18) « x « " == " « xx « std::endl;
}
```

10.3 examples/fp16.cu

How to leverage CUDA __half and __half2 16 bits floating points number using aerobus::i16 Warning : due to an NVIDIA bug (lack of constexpr operators), performance is not good

```
// TO compile with nvcc -03 -std=c++20 -arch=sm_90 fp16.cu
#include <cstdio>
#define WITH_CUDA_FP16
#include "../src/aerobus.h"
constexpr size_t N = 1 « 24;
change int_type to aerobus::i32 (or i64) and float_type to float (resp. double)
to see how good is the generated assembly compared to what nvcc generates for 16 bits
using int_type = aerobus::i16;
using float_type = __half2;
float rand(float min, float max) {
  float range = (max - min);
  float div = RAND_MAX / range;
return min + (rand() / div); // NOLINT
using EXPM1 = aerobus::expm1<int_type, 6>;
__device__ INLINED float_type f(float_type x) {
    return EXPM1::eval(x);
 _global__ void run(size_t N, float_type* in, float_type* out) {
   for(size_t i = threadIdx.x + blockDim.x * blockIdx.x; i < N; i += blockDim.x * gridDim.x) {
         out[i] = f(f(f(f(f(in[i])))));
}
int main() {
     __half2 *d_in, *d_out;
     cudaMalloc<__half2>(&d_in, N * sizeof(__half2));
     cudaMalloc<__half2>(&d_out, N * sizeof(__half2));
    __half2 *in = reinterpret_cast<__half2*>(malloc(N * sizeof(__half2)));
__half2 *out = reinterpret_cast<__half2*>(malloc(N * sizeof(__half2)));
     for(size_t i = 0; i < N; ++i) +
         in[i] = \underline{\quad} half2(\underline{\quad} float2half(rand(-0.01, 0.01)), \underline{\quad} float2half(rand(-0.01, 0.01)));
    cudaMemcpy(d in, in, N * sizeof( half2), cudaMemcpyHostToDevice);
     run«<128, 512»>(N, d_in, d_out);
     cudaMemcpy(out, d_out, N * sizeof(__half2), cudaMemcpyDeviceToHost);
     cudaFree(d in);
     cudaFree(d_out);
```

10.4 examples/continued_fractions.cpp

How to use aerobus::ContinuedFraction to get approximations of known numbers

```
#include <cmath>
#include <iostream>
```

10.5 examples/modular_arithmetic.cpp

How to use aerobus::zpz to perform computations on rational fractions with coefficients in modular rings

```
#include <iostream>
#include "../src/aerobus.h"

using FIELD = aerobus::zpz<2>;
using POLYNOMIALS = aerobus::polynomial<FIELD>;
using FRACTIONS = aerobus::FractionField<POLYNOMIALS>;

// x^3 + 2x^2 + 1, with coefficients in Z/2Z, actually x^3 + 1
using P = aerobus::make_int_polynomial_t<FIELD, 1, 2, 0, 1>;

// x^3 + 5x^2 + 7x + 11 with coefficients in Z/17Z, meaning actually x^3 + x^2 + 1
using Q = aerobus::make_int_polynomial_t<FIELD, 1, 5, 8, 1>;

// P/Q in the field of fractions of polynomials
using F = aerobus::makefraction_t<POLYNOMIALS, P, Q>;

int main() {
   const double v = F::eval<double>(1.0);
   std::cout « "expected = " « 2.0/3.0 « std::endl;
   std::cout « "value = " « v « std::endl;
   return 0;
}
```

10.6 examples/make_polynomial.cpp

```
How to build your own sequence of known polynomials, here    Abel polynomials
#include <iostream>
#include "../src/aerobus.h"

// let's build Abel polynomials from scratch using Aerobus
// note : it's now integrated in the main library, but still serves as an example
template<typename I = aerobus::i64>
```

```
struct AbelHelper {
private:
    using P = aerobus::polynomial<I>;
    // to keep recursion working, we need to operate on a\!*\!n and not just a
    template<size_t deg, I::inner_type an>
    struct Inner {
    // abel(n, a) = (x-an) * abel(n-1, a)
    using type = typename aerobus::mul_t<</pre>
            typename Inner<deg-1, an>::type,
             typename aerobus::sub_t<typename P::X, typename P::template inject_constant_t<an>>
    };
    // abel(0, a) = 1
    template<I::inner_type an>
    struct Inner<0, an> {
        using type = P::one;
    // abel(1, a) = X
    template<I::inner_type an>
    struct Inner<1, an> {
```

208 Examples

```
using type = P::X;
};

template<size_t n, auto a, typename I = aerobus::i64>
using AbelPolynomials = typename AbelHelper<I>::template Inner<n, a*n>::type;

using A2_3 = AbelPolynomials<3, 2>;

int main() {
    std::cout « "expected = x^3 - 12 x^2 + 36 x" « std::endl;
    std::cout « "aerobus = " « A2_3::to_string() « std::endl;
    return 0;
}
```

10.7 examples/polynomials over finite field.cpp

How to build a known polynomial (here aerobus::known_polynomials::allone) with coefficients in a finite field (here aerobus::zpz<2>) and get its value when evaluated at a value in this field (here 1).

```
#include <iostream>
#include "../src/aerobus.h"

using GF2 = aerobus::zpz<2>;
using P = aerobus::known_polynomials::allone<8, GF2>;

int main() {
    // at this point, value_at_1 is an instanciation of zpz<2>::val
    using value_at_1 = P::template value_at_t<GF2::template inject_constant_t<1»;
    // here we get its value in an arithmetic type, here int32_t
    constexpr int32_t x = value_at_1::template get<int32_t>();
    // ensure that l+l+l+l+l+l+l in Z/2Z is equal to one
    std::cout « "expected = " « 1 « std::endl;
    std::cout « "computed = " « x « std::endl;
    return 0;
}
```

10.8 examples/compensated_horner.cpp

How to use compensated horner evaluation scheme to get better accuracy when evaluating polynomials close to its roots

See also

```
publication
```

```
// run with ./generate_comp_horner.sh in this directory
// that will compile and run this sample and plot all the generated data
#include "../src/aerobus.h'
using namespace aerobus; // NOLINT
constexpr size_t NB_POINTS = 400;
template<typename P, typename T, bool compensated>
DEVICE INLINED T eval(const T& x) {
   if constexpr (compensated) {
        return P::template compensated_eval<T>(x);
    } else {
        return P::template eval<T>(x);
template<typename T>
DEVICE INLINED T exact_large(const T& x) {
    return pow_scalar<T, 5>(0.75 - x) * pow_scalar<T, 11>(1 - x);
template<typename T>
DEVICE INLINED T exact_small(const T& x) {
    return pow_scalar<T, 3>(x - 1);
```

```
}
template<typename P, typename T, bool compensated>
void run(T left, T right, const char *file_name, T (*exact)(const T&)) {
   FILE *f = ::fopen(file_name, "w+");
   T step = (right - left) / NB_POINTS;
     T x = left;
      for (size_t i = 0; i <= NB_POINTS; ++i) {</pre>
          ::fprintf(f, "%e %e %e\n", x, eval<P, T, compensated>(x), exact(x));
          x += step;
     ::fclose(f);
}
int main() {
           // (0.75 - x)^5 * (1 - x)^11
          using P = mul_t<
               pow_t<pq64, pq64::val<
                     typename q64::template inject_constant_t<-1>,
                     q64::val<i64::val<3>, i64::val<4>», 5>,
               pow_t<pq64. pq64::val<typename q64::template inject_constant_t<-1>, typename q64::one>, 11>
          >;
          vising FLOAT = double;
run<P, FLOAT, false>(0.68, 1.15, "plots/large_sample_horner.dat", &exact_large);
run<P, FLOAT, true>(0.68, 1.15, "plots/large_sample_comp_horner.dat", &exact_large);
          run<P, FLOAT, false>(0.74995, 0.75005, "plots/first_root_horner.dat", &exact_large);
run<P, FLOAT, true>(0.74995, 0.75005, "plots/first_root_comp_horner.dat", &exact_large);
          run<P, FLOAT, false>(0.9935, 1.0065, "plots/second_root_horner.dat", &exact_large);
run<P, FLOAT, true>(0.9935, 1.0065, "plots/second_root_comp_horner.dat", &exact_large);
           // (x - 1) ^ 3
          using P = make_int_polynomial_t<i32, 1, -3, 3, -1>;
          run<P, double, false>(1-0.00005, 1+0.00005, "plots/double.dat", &exact_small);
           run<P, float, true>(1-0.00005, 1+0.00005, "plots/float_comp.dat", &exact_small);
}
```

210 Examples

Index

```
abs t
                                                             mulfractions t, 29
     aerobus, 20
                                                             pi64, 30
add t
                                                             PI fraction, 30
    aerobus, 20
                                                             pow t, 30
    aerobus::i32, 57
                                                             pq64, 30
    aerobus::i64, 63
                                                             q32, 30
    aerobus::polynomial < Ring >, 72
                                                             q64, 31
    aerobus::Quotient< Ring, X >, 80
                                                             sin, 31
    aerobus::zpz , 105
                                                             sinh, 31
                                                             SQRT2 fraction, 31
addfractions t
    aerobus, 20
                                                             SQRT3 fraction, 31
aerobus, 15
                                                             stirling_1_signed_t, 32
    abs_t, 20
                                                             stirling_1_unsigned_t, 32
    add_t, 20
                                                             stirling_2_t, 32
    addfractions t, 20
                                                             sub t, 33
    aligned_malloc, 34
                                                             tan, 33
    alternate_t, 20
                                                             tanh, 33
    alternate_v, 35
                                                             taylor, 33
    asin, 21
                                                             vadd t, 34
    asinh, 21
                                                             vmul_t, 34
                                                         aerobus::ContinuedFraction < a0 >, 45
    atan, 21
    atanh, 21
                                                             type, 45
    bell t, 23
                                                             val, 46
    bernoulli t, 23
                                                         aerobus::ContinuedFraction < a0, rest... >, 46
    bernoulli v, 35
                                                             type, 47
    combination t, 23
                                                             val, 47
    combination v, 35
                                                         aerobus::ContinuedFraction < values >, 44
                                                         aerobus::ConwayPolynomial, 47
    cos, 23
    cosh, 25
                                                         aerobus::Embed< i32, i64 >, 49
    div t, 25
                                                             type, 49
    E fraction, 25
                                                         aerobus::Embed< polynomial< Small >, polynomial<
    embed_int_poly_in_fractions_t, 25
                                                                  Large >>, 50
    exp, 26
                                                             type, 50
    expm1, 26
                                                         aerobus::Embed< q32, q64 >, 51
    factorial t, 26
                                                             type, 51
    factorial_v, 35
                                                         aerobus::Embed< Quotient< Ring, X >, Ring >, 52
    field, 34
    fpq32, 26
                                                         aerobus::Embed< Ring, FractionField< Ring >>, 53
    fpq64, 27
                                                             type, 53
                                                         aerobus::Embed< Small, Large, E >, 49
    FractionField, 27
    gcd_t, 27
                                                         aerobus::Embed< zpz< x>, i32>, 53
    geometric sum, 27
                                                             type, 54
    Inp1, 27
                                                         aerobus::i32, 55
                                                             add t, 57
    make_frac_polynomial_t, 28
    make int polynomial t, 28
                                                             div t, 57
    make q32 t, 28
                                                             eq_t, 57
    make_q64_t, 29
                                                             eq_v, 60
    makefraction_t, 29
                                                             gcd_t, 57
    mul t, 29
                                                             gt_t, 58
```

inject_constant_t, 58	aerobus::known_polynomials, 40
inject_ring_t, 58	hermite_kind, 40
inner_type, 58	physicist, 40
is_euclidean_domain, 60	probabilist, 40
is_field, 60	aerobus::polynomial< Ring >, 70
It_t, 58	add_t, 72
mod_t, 59	derive_t, 72
mul_t, 59	div_t, 72
one, 59	eq_t, 73
pos_t, 59	gcd_t, 73
pos_v, 60	gt_t, 73
sub_t, 60	inject_constant_t, 74
zero, 60	inject_ring_t, 74
aerobus::i32::val $< x >$, 89	is_euclidean_domain, 78
enclosing_type, 90	is_field, 78
get, 90	lt_t, 74
is_zero_t, 90	mod_t, 74
to_string, 90	monomial_t, 75
v, 90	mul_t, 75
aerobus::i64, 62	one, 75
add_t, 63	pos_t, 75
	pos v, 78
div_t, 64	• —
eq_t, 64	simplify_t, 77
eq_v, 67	sub_t, 77
gcd_t, 64	X, 77
gt_t, 64	zero, 77
gt_v, 67	aerobus::polynomial< Ring >::compensated_horner<
inject_constant_t, 65	arithmeticType, $P > ::EFTHorner < index,$
inject_ring_t, 65	ghost $>$, 47
inner_type, 65	func, 48
is_euclidean_domain, 67	aerobus::polynomial< Ring >::compensated_horner<
is_field, 67	arithmeticType, P >::EFTHorner<-1, ghost >,
lt_t, 65	48
It v, 68	func, 48
mod_t, 66	aerobus::polynomial< Ring >::horner_reduction_t< P
mul_t, 66	>, 54
one, 66	aerobus::polynomial< Ring >::horner_reduction_t< P
pos_t, 66	>::inner< index, stop >, 68
pos_v, 68	type, 69
• —	
sub_t, 66	aerobus::polynomial < Ring >::horner_reduction_t < P
zero, 67	>::inner< stop, stop >, 69
aerobus::i64::val< x >, 91	type, 69
enclosing_type, 92	aerobus::polynomial < Ring >::val < coeffN >, 100
get, 92	aN, 101
inner_type, 92	coeff_at_t, 101
is_zero_t, 92	compensated_eval, 102
to_string, 92	degree, 103
v, 93	enclosing_type, 101
aerobus::internal, 36	eval, 102
index_sequence_reverse, 40	is_zero_t, 101
is_instantiation_of_v, 40	is_zero_v, 103
make_index_sequence_reverse, 39	ring_type, 102
type_at_t, 39	strip, 102
aerobus::is_prime $<$ n $>$, 69	to_string, 102
value, 70	value_at_t, 102
aerobus::IsEuclideanDomain, 41	aerobus::polynomial< Ring >::val< coeffN >::coeff_at<
aerobus::IsField, 41	aeropusporynomiai< miny /vai< CUEIIN /CUEII al<
ARTHUR ISCIPIT 41	
aerobus::IsRing, 42	index, E >, 43 aerobus::polynomial < Ring >::val < coeffN >::coeff_at <

```
index, std::enable_if_t<(index< 0 | | index >
                                                               add_t, 105
                                                               div t, 105
          0)>>, 43
     type, 43
                                                               eq_t, 106
aerobus::polynomial < Ring >::val < coeffN >::coeff_at <
                                                               eq_v, 109
          index, std::enable_if_t<(index==0)>>, 44
                                                               gcd_t, 106
                                                               gt t, 106
aerobus::polynomial< Ring >::val< coeffN, coeffs >,
                                                               gt_v, 109
          93
                                                               inject_constant_t, 107
     aN, 94
                                                               inner type, 107
     coeff_at_t, 94
                                                               is euclidean domain, 109
     compensated_eval, 95
                                                               is field, 109
     degree, 97
                                                               lt_t, 107
     enclosing_type, 94
                                                               It_v, 109
     eval, 96
                                                               mod_t, 107
                                                               mul_t, 107
     is_zero_t, 95
     is_zero_v, 97
                                                               one, 108
     ring type, 95
                                                               pos t, 108
     strip, 95
                                                               pos v, 110
     to_string, 96
                                                               sub_t, 108
     value_at_t, 95
                                                               zero, 108
aerobus::Quotient< Ring, X >, 79
                                                          aerobus::zpz<p>::val<math><x>, 98
     add t, 80
                                                               enclosing type, 99
                                                               get, 99
     div_t, 81
     eq_t, 81
                                                               is_zero_t, 99
     eq_v, 83
                                                               is zero v, 100
                                                               to_string, 99
     inject_constant_t, 81
     inject_ring_t, 81
                                                               v, 100
     is euclidean domain, 83
                                                          aligned malloc
     mod t, 82
                                                               aerobus, 34
     mul t, 82
                                                          alternate t
     one, 82
                                                               aerobus, 20
     pos_t, 82
                                                          alternate_v
     pos v, 83
                                                               aerobus, 35
                                                          aN
     zero, 83
aerobus::Quotient < Ring, X >::val < V >, 97
                                                               aerobus::polynomial < Ring >::val < coeffN >, 101
     raw_t, 98
                                                               aerobus::polynomial< Ring >::val< coeffN, coeffs
     type, 98
                                                                    >, 94
aerobus::type_list< Ts >, 85
                                                          asin
     at, 86
                                                               aerobus, 21
     concat, 86
                                                          asinh
     insert, 86
                                                               aerobus, 21
     length, 87
                                                          at
     push back, 86
                                                               aerobus::type_list< Ts >, 86
     push front, 87
                                                          atan
     remove, 87
                                                               aerobus, 21
aerobus::type_list< Ts >::pop_front, 78
                                                          atanh
     tail, 79
                                                               aerobus, 21
     type, 79
                                                          bell t
aerobus::type_list< Ts >::split< index >, 84
                                                               aerobus, 23
     head, 84
                                                          bernoulli t
     tail, 84
                                                               aerobus, 23
aerobus::type list<>, 88
                                                          bernoulli v
     concat, 88
                                                               aerobus, 35
     insert, 88
     length, 89
                                                          coeff at t
     push_back, 88
                                                               aerobus::polynomial < Ring >::val < coeffN >, 101
     push_front, 88
                                                               aerobus::polynomial< Ring >::val< coeffN, coeffs
aerobus::zpz , 103
                                                                    >, <mark>94</mark>
```

```
combination_t
                                                            aerobus, 26
     aerobus, 23
                                                       factorial_t
combination v
                                                            aerobus, 26
    aerobus, 35
                                                       factorial v
compensated eval
                                                            aerobus, 35
     aerobus::polynomial < Ring >::val < coeffN >, 102
                                                       field
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                            aerobus, 34
          >, 95
                                                       fpq32
concat
                                                            aerobus, 26
     aerobus::type list< Ts >, 86
                                                       fpq64
     aerobus::type_list<>, 88
                                                            aerobus, 27
cos
                                                       FractionField
     aerobus, 23
                                                            aerobus, 27
cosh
     aerobus, 25
                                                       func
                                                            aerobus::polynomial < Ring >::compensated horner <
                                                                 arithmeticType, P >::EFTHorner< index,
degree
     aerobus::polynomial < Ring >::val < coeffN >, 103
                                                                 ghost >, 48
                                                            aerobus::polynomial < Ring >::compensated_horner <
     aerobus::polynomial < Ring >::val < coeffN, coeffs
                                                                 arithmeticType, P >::EFTHorner <- 1, ghost >,
derive t
     aerobus::polynomial < Ring >, 72
                                                       gcd t
div t
                                                            aerobus, 27
     aerobus, 25
                                                            aerobus::i32, 57
     aerobus::i32, 57
                                                            aerobus::i64, 64
     aerobus::i64, 64
                                                            aerobus::polynomial< Ring >, 73
     aerobus::polynomial < Ring >, 72
                                                            aerobus::zpz< p>, 106
     aerobus::Quotient< Ring, X >, 81
                                                        geometric_sum
     aerobus::zpz , 105
                                                            aerobus, 27
E fraction
                                                       get
     aerobus, 25
                                                            aerobus::i32::val< x >, 90
embed_int_poly_in_fractions_t
                                                            aerobus::i64::val < x >, 92
    aerobus, 25
                                                            aerobus::zpz ::val < x >, 99
enclosing type
                                                       gt_t
     aerobus::i32::val< x >, 90
                                                            aerobus::i32, 58
     aerobus::i64::val < x >, 92
                                                            aerobus::i64, 64
     aerobus::polynomial < Ring >::val < coeffN >, 101
                                                            aerobus::polynomial < Ring >, 73
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                            aerobus::zpz , 106
          >, 94
                                                       gt_v
     aerobus::zpz ::val < x >, 99
                                                            aerobus::i64, 67
eq t
                                                            aerobus::zpz, 109
     aerobus::i32, 57
                                                       head
     aerobus::i64, 64
                                                            aerobus::type list< Ts >::split< index >, 84
     aerobus::polynomial < Ring >, 73
     aerobus::Quotient < Ring, X >, 81
                                                       hermite kind
                                                            aerobus::known_polynomials, 40
     aerobus::zpz, 106
eq_v
                                                       index_sequence_reverse
     aerobus::i32, 60
                                                            aerobus::internal, 40
     aerobus::i64, 67
                                                       inject constant t
     aerobus::Quotient< Ring, X >, 83
                                                            aerobus::i32, 58
     aerobus::zpz , 109
                                                            aerobus::i64, 65
eval
                                                            aerobus::polynomial < Ring >, 74
     aerobus::polynomial < Ring >::val < coeffN >, 102
                                                            aerobus::Quotient < Ring, X >, 81
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                            aerobus::zpz , 107
         >, 96
                                                       inject_ring_t
exp
                                                            aerobus::i32, 58
    aerobus, 26
                                                            aerobus::i64, 65
expm1
```

aerobus::polynomial< Ring >, 74	make_q64_t
aerobus::Quotient< Ring, X >, 81	aerobus, 29
inner_type	makefraction_t
aerobus::i32, 58	aerobus, 29
aerobus::i64, 65	mod_t
aerobus::i64::val < $x >$, 92	aerobus::i32, 59
aerobus:: $zpz $, 107	aerobus::i64, 66
insert	aerobus::polynomial< Ring >, 74
aerobus::type_list< Ts >, 86	aerobus::Quotient< Ring, X >, 82
aerobus::type_list<>, 88	aerobus::zpz, 107
Introduction, 1	monomial t
is_euclidean_domain	aerobus::polynomial< Ring >, 75
aerobus::i32, 60	mul_t
aerobus::i64, 67	aerobus, 29
aerobus::polynomial< Ring >, 78	aerobus::i32, 59
aerobus::Quotient< Ring, X >, 83	aerobus::i64, 66
aerobus::zpz, 109	aerobus::polynomial< Ring >, 75
is field	aerobus::Quotient< Ring, X >, 82
aerobus::i32, 60	aerobus::zpz $<$ p $>$, 107
aerobus::i64, 67	mulfractions_t
aerobus::polynomial < Ring >, 78	aerobus, 29
aerobus:: $zpz $, 109	aerobus, 25
is_instantiation_of_v	one
	aerobus::i32, 59
aerobus::internal, 40	aerobus::i64, 66
is_zero_t	aerobus::polynomial< Ring >, 75
aerobus::i32::val< x >, 90	aerobus::Quotient< Ring, X >, 82
aerobus::i64::val< x >, 92	
aerobus::polynomial< Ring >::val< coeffN >, 101	aerobus::zpz, 108
aerobus::polynomial < Ring >::val < coeffN, coeffs	physicist
>, 95	aerobus::known_polynomials, 40
aerobus:: $zpz $:: $val < x >$, 99	pi64
is_zero_v	•
aerobus::polynomial< Ring >::val< coeffN >, 103	aerobus, 30
aerobus::polynomial< Ring >::val< coeffN, coeffs	PI_fraction
>, 97	aerobus, 30
aerobus::zpz $<$ p $>$::val $<$ x $>$, 100	pos_t
	aerobus::i32, 59
length	aerobus::i64, 66
aerobus::type_list< Ts >, 87	aerobus::polynomial< Ring >, 75
aerobus::type_list<>, 89	aerobus::Quotient< Ring, X >, 82
Inp1	aerobus::zpz, 108
aerobus, 27	pos_v
lt_t	aerobus::i32, 60
aerobus::i32, 58	aerobus::i64, 68
aerobus::i64, 65	aerobus::polynomial $<$ Ring $>$, 78
aerobus::polynomial < Ring >, 74	aerobus::Quotient $<$ Ring, X $>$, 83
aerobus::zpz, 107	aerobus::zpz $<$ p $>$, 110
lt_v	pow_t
aerobus::i64, 68	aerobus, 30
aerobus:: $zpz $, 109	pq64
г. г. г. г.	aerobus, 30
make_frac_polynomial_t	probabilist
aerobus, 28	aerobus::known_polynomials, 40
make_index_sequence_reverse	push_back
aerobus::internal, 39	aerobus::type_list< Ts >, 86
make_int_polynomial_t	aerobus::type_list<>, 88
aerobus, 28	push_front
make_q32_t	aerobus::type_list< Ts >, 87
aerobus, 28	aerobus::type_list<>, 88
	

q32	aerobus::zpz $<$ p $>$::val $<$ x $>$, 99
aerobus, 30	type
q64	aerobus::ContinuedFraction< a0 >, 45
aerobus, 31	aerobus::ContinuedFraction < a0, rest >, 47
raw t	aerobus::Embed< i32, i64 >, 49
raw_t aerobus::Quotient< Ring, X >::val< V >, 98	aerobus::Embed< polynomial< Small >,
README.md, 111	polynomial < Large > >, 50
remove	aerobus::Embed< q32, q64 >, 51 aerobus::Embed< Quotient< Ring, X >, Ring >,
aerobus::type_list< Ts >, 87	52
ring_type	aerobus::Embed< Ring, FractionField< Ring > >,
aerobus::polynomial< Ring >::val< coeffN >, 102	53
aerobus::polynomial< Ring >::val< coeffN, coeffs	aerobus::Embed $<$ zpz $<$ x $>$, i32 $>$, 54
>, 95	aerobus::polynomial < Ring >::horner_reduction_t <
· Pr	P >::inner< index, stop >, 69
simplify_t	$aerobus::polynomial < Ring > ::horner_reduction_t <$
aerobus::polynomial < Ring >, 77	P >::inner< stop, stop >, 69
aerobus, 31	aerobus::polynomial< Ring >::val< coeffN
sinh	>::coeff_at< index, std::enable_if_t<(index<
aerobus, 31	0 index > 0)> >, 43
SQRT2_fraction	aerobus::polynomial< Ring >::val< coeffN
aerobus, 31	>::coeff_at< index, std::enable_if_t<(index==0)>
SQRT3_fraction	>, 44
aerobus, 31	aerobus::Quotient< Ring, X >::val< V >, 98 aerobus::type_list< Ts >::pop_front, 79
src/aerobus.h, 111	type_at_t
src/examples.h, 204	aerobus::internal, 39
stirling_1_signed_t	as: 3235:
aerobus, 32	V
stirling_1_unsigned_t	aerobus::i32::val $< x >$, 90
aerobus, 32	aerobus::i64::val< x >, 93
stirling_2_t	aerobus::zpz $<$ p $>$::val $<$ x $>$, 100
aerobus, 32	vadd_t
strip aerobus::polynomial < Ring >::val < coeffN >, 102	aerobus, 34
aerobus::polynomial< Ring >::val< coeffN, coeffs	val
>, 95	aerobus::ContinuedFraction< a0 >, 46 aerobus::ContinuedFraction< a0, rest >, 47
sub t	value
aerobus, 33	aerobus::is_prime< n >, 70
aerobus::i32, 60	value_at_t
aerobus::i64, 66	aerobus::polynomial< Ring >::val< coeffN >, 102
aerobus::polynomial $<$ Ring $>$, 77	aerobus::polynomial< Ring >::val< coeffN, coeffs
aerobus::zpz, 108	>, 95
tail	vmul_t
tail aerobus::type_list< Ts >::pop_front, 79	aerobus, 34
aerobus::type_list< Ts >::pop_nont, 79 aerobus::type_list< Ts >::split< index >, 84	X
tan	aerobus::polynomial< Ring >, 77
aerobus, 33	acrobuspolynomial \ Timy >, 77
tanh	zero
aerobus, 33	aerobus::i32, 60
taylor	aerobus::i64, 67
aerobus, 33	aerobus::polynomial< Ring >, 77
to_string	aerobus::Quotient $<$ Ring, X $>$, 83
aerobus::i32::val $< x >$, 90	aerobus::zpz, 108
aerobus::i64::val< x >, 92	
aerobus::polynomial < Ring >::val < coeffN >, 102	
aerobus::polynomial < Ring >::val < coeffN, coeffs	
>, 96	