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
$8.12 \; aerobus:: Embed < polynomial < Small >, polynomial < Large >> Struct \; Template \; Reference \; \ldots \; 5 \; description \; Struct \; Stru$	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
$8.17 \ aerobus::polynomial < Ring > ::horner_reduction_t < P > Struct \ Template \ Reference \ . \ . \ . \ . \ . \ .$	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	. 204
10 Examples	205
10.1 examples/hermite.cpp	. 205
10.2 examples/custom_taylor.cpp	
10.3 examples/fp16.cu	. 206
10.4 examples/continued_fractions.cpp	
10.5 examples/modular_arithmetic.cpp	. 207
10.6 examples/make_polynomial.cpp	
10.7 examples/polynomials_over_finite_field.cpp	
10.8 examples/compensated_horner.cpp	
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></large></small>	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

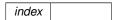


8.27.2.6 remove

```
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]
```

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.
```

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 note this makes no sense on integer types as arithmetic on integers is exact in IEEE

WARNING: this does not work with gcc with -O3 optimization level because gcc does illegal stuff with floating point arithmetic

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 & 0x7fffffU);
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;
                    T z = *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;
#ifdef __clang_
00321
00322
                    *y = fma_helper<T>::eval(a, b, -*x);
00323
                    #else
00324
                    T ah, al, bh, bl;
                    split(a, &ah, &al);
split(b, &bh, &bl);
*y = al * bl - (((*x - ah * bh) - al * bh) - ah * bl);
00325
00326
00327
00328
00329
                }
00330
               template<typename T, size_t N>
static INLINED DEVICE T horner(T *p1, T *p2, T x) {
00331
00332
                    T r = p1[0] + p2[0];
00333
00334
                    for (int64_t i = N - 1; i >= 0; --i) {
00335
                        r = r * x + p1[N - i] + p2[N - i];
00336
00337
00338
                    return r:
00339
              // namespace internal
00340
00341 } // namespace aerobus
00342
00343 // utilities
00344 namespace aerobus {
00345
         namespace internal {
               template<template<typename...> typename TT, typename T>
00347
                struct is_instantiation_of : std::false_type { };
00348
               template<template<typename...> typename TT, typename... Ts>
struct is_instantiation_of<TT, TT<Ts...» : std::true_type { };</pre>
00349
00350
00351
                template<template<typename...> typename TT, typename T>
00353
                inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00354
00355
                template <int64_t i, typename T, typename... Ts>
00356
                struct type_at {
                   static_assert(i < sizeof...(Ts) + 1, "index out of range");
using type = typename type_at<i - 1, Ts...>::type;
00357
00358
00359
00360
00361
                template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00362
                    using type = T;
00363
                };
00364
00365
                template <size_t i, typename... Ts>
00366
                using type_at_t = typename type_at<i, Ts...>::type;
00367
00368
00369
                template<size_t n, size_t i, typename E = void>
00370
               struct _is_prime {};
00371
00372
                template<size_t i>
00373
                struct _is_prime<0, i> {
                    static constexpr bool value = false;
00374
00375
00376
00377
                template<size_t i>
00378
                struct _is_prime<1, i> {
00379
                    static constexpr bool value = false;
00380
00381
00382
               template<size t i>
```

```
struct _is_prime<2, i> {
00384
                 static constexpr bool value = true;
00385
              };
00386
00387
              template<size t i>
00388
              struct _is_prime<3, i> {
                 static constexpr bool value = true;
00389
00390
00391
00392
              template<size_t i>
              struct _is_prime<5, i> {
00393
                 static constexpr bool value = true;
00394
00395
00396
00397
              template<size_t i>
00398
              struct _is_prime<7, i> {
                  static constexpr bool value = true;
00399
00400
00401
00402
              template<size_t n, size_t i>
00403
              struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)» {
00404
                  static constexpr bool value = false;
00405
              };
00406
00407
              template<size_t n, size_t i>
              struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)» {
00408
00409
                  static constexpr bool value = false;
00410
00411
00412
              {\tt template} < {\tt size\_t n, size\_t i} >
              struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)» {
00413
00414
                  static constexpr bool value = true;
00415
00416
00417
              {\tt template}{<} {\tt size\_t n, size\_t i}{\gt}
              struct _is_prime<n, i, std::enable_if_t<(
00418
00419
                 n % i == 0 &&
                  n >= 9 &&
00421
                  n % 3 != 0 &&
00422
                  n % 2 != 0 &&
00423
                  i * i > n)» {
00424
                  static constexpr bool value = true;
00425
              }:
00426
00427
              template<size_t n, size_t i>
00428
              struct _is_prime<n, i, std::enable_if_t<(
00429
                 n % (i+2) == 0 &&
00430
                  n >= 9 &&
                  n % 3 != 0 &&
00431
                  n % 2 != 0 &&
00432
00433
                  i * i <= n) » {
00434
                  static constexpr bool value = true;
00435
00436
00437
              template<size_t n, size_t i>
              struct _is_prime<n, i, std::enable_if_t<(
    n % (i+2) != 0 &&</pre>
00438
00439
00440
                      n % i != 0 &&
00441
                       n >= 9 &&
00442
                      n % 3 != 0 &&
                      n % 2 != 0 &&
00443
00444
                       (i * i <= n)) > {
00445
                  static constexpr bool value = _is_prime<n, i+6>::value;
00446
00447
          } // namespace internal
00448
00451
          template<size_t n>
00452
          struct is_prime {
00454
              static constexpr bool value = internal::_is_prime<n, 5>::value;
00455
00456
00460
          template<size_t n>
00461
          static constexpr bool is_prime_v = is_prime<n>::value;
00462
00463
          // gcd
00464
          namespace internal {
00465
              template <std::size_t... Is>
00466
              constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
00467
                   -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
00468
00469
              template <std::size t N>
00470
              using make_index_sequence_reverse
00471
                   = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00472
00478
              template<typename Ring, typename E = void>
00479
              struct gcd;
00480
```

```
00481
               template<typename Ring>
00482
               struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
00483
                   template<typename A, typename B, typename E = void>
00484
                   struct gcd_helper {};
00485
00486
                   // B = 0, A > 0
                   template<typename A, typename B>
00487
00488
                   struct gcd_helper<A, B, std::enable_if_t<
00489
                      ((B::is_zero_t::value) &&
00490
                           (Ring::template gt_t<A, typename Ring::zero>::value))» {
00491
                       using type = A;
00492
                   };
00493
00494
                   // B = 0, A < 0
                   template<typename A, typename B>
00495
00496
                   struct gcd_helper<A, B, std::enable_if_t<
00497
                       ((B::is_zero_t::value) &&
                       !(Ring::template gt_t<A, typename Ring::zero>::value))» {
using type = typename Ring::template sub_t<typename Ring::zero, A>;
00498
00499
00500
                   } ;
00501
00502
                   // B != 0
                   template<typename A, typename B> \,
00503
                   struct gcd_helper<A, B, std::enable_if_t<
00504
00505
                       (!B::is_zero_t::value)
00506
00507
                   private: // NOLINT
                       // A / B using k = typename Ring::template div_t<A, B>;
00508
00509
00510
                       // A - (A/B) *B = A % B
00511
                       using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
00512
00513
                   public:
00514
                       using type = typename gcd_helper<B, m>::type;
00515
                   };
00516
                   template<typename A, typename B>
using type = typename gcd_helper<A, B>::type;
00517
00519
               };
00520
          } // namespace internal
00521
          // yadd and ymul
00522
00523
          namespace internal {
00524
              template<typename... vals>
00525
              struct vmul {};
00526
00527
              template<typename v1, typename... vals>
              struct vmul<v1, vals...> {
00528
                  using type = typename v1::enclosing_type::template mul_t<v1, typename
00529
     vmul<vals...>::tvpe>;
00530
              };
00531
00532
              template<typename v1>
              struct vmul<v1> {
00533
00534
                   using type = v1;
00535
              };
00536
00537
              template<typename... vals>
00538
              struct vadd {};
00539
00540
              template<typename v1, typename... vals>
              struct vadd<v1, vals...> {
00541
00542
                 using type = typename v1::enclosing_type::template add_t<v1, typename
      vadd<vals...>::type>;
00543
             };
00544
00545
              template<typename v1>
00546
              struct vadd<v1> {
00547
                  using type = v1;
00548
               };
00549
          } // namespace internal
00550
00553
          template<typename T, typename A, typename B>
          using gcd_t = typename internal::gcd<T>::template type<A, B>;
00554
00555
00559
          template<typename... vals>
00560
          using vadd_t = typename internal::vadd<vals...>::type;
00561
00565
          template<typename... vals>
          using vmul_t = typename internal::vmul<vals...>::type;
00566
00567
00571
          template<typename val>
00572
          requires IsEuclideanDomain<typename val::enclosing_type>
00573
          using abs_t = std::conditional_t<
00574
                           val::enclosing_type::template pos_v<val>,
      val, typename val::enclosing_type::template
sub_t<typename val::enclosing_type::zero, val>>;
00575
```

```
00576 } // namespace aerobus
00577
00578 // embedding
00579 namespace aerobus {
00584
         template<typename Small, typename Large, typename E = void>
00585
          struct Embed:
00586 } // namespace aerobus
00587
00588 namespace aerobus {
          template<typename Ring, typename X>
00593
00594
          requires IsRing<Ring>
00595
          struct Ouotient {
00598
              template <typename V>
00599
              struct val {
00600
              public:
00601
                  using raw_t = V;
00602
                  using type = abs_t<typename Ring::template mod_t<V, X>>;
00603
              };
00604
00606
              using zero = val<typename Ring::zero>;
00607
00609
              using one = val<typename Ring::one>;
00610
00614
              template<typename v1, typename v2>
00615
              using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00616
00620
              template<typename v1, typename v2>
00621
              using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00622
00626
              template<typename v1, typename v2>
00627
              using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00628
00632
              template<typename v1, typename v2>
00633
              using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00634
              template<typename v1, typename v2>
00638
00639
              using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00640
00644
              template<typename v1, typename v2>
00645
              static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00646
              template<typename v1>
00650
00651
              using pos_t = std::true_type;
00652
00656
              template<typename v>
00657
              static constexpr bool pos_v = pos_t<v>::value;
00658
00660
              static constexpr bool is_euclidean_domain = true;
00661
00665
              template<auto x>
00666
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00667
00671
              template < typename v >
00672
              using inject_ring_t = val<v>;
00673
          };
00674
00678
          template<typename Ring, typename X>
          struct Embed<Quotient<Ring, X>, Ring> {
00679
00682
              template<typename val>
00683
              using type = typename val::raw_t;
00684
00685 }
         // namespace aerobus
00686
00687 // type_list
00688 namespace aerobus {
00690
          template <typename... Ts>
00691
          struct type_list;
00692
00693
          namespace internal {
00694
              template <typename T, typename... Us>
00695
              struct pop_front_h {
                  using tail = type_list<Us...>;
using head = T;
00696
00697
00698
              };
00699
00700
              template <size_t index, typename L1, typename L2>
              struct split_h {
00701
00702
               private:
                  static_assert(index <= L2::length, "index ouf of bounds");</pre>
00703
00704
                  using a = typename L2::pop_front::type;
00705
                  using b = typename L2::pop_front::tail;
00706
                  using c = typename L1::template push_back<a>;
00707
               public:
00708
                  using head = typename split_h<index - 1, c, b>::head;
using tail = typename split_h<index - 1, c, b>::tail;
00709
00710
00711
              };
```

```
00712
00713
              template <typename L1, typename L2>
00714
              struct split_h<0, L1, L2> {
                using head = L1;
using tail = L2;
00715
00716
00717
              };
00718
00719
               template <size_t index, typename L, typename T>
00720
              struct insert_h {
                   static_assert(index <= L::length, "index ouf of bounds");</pre>
00721
                   using s = typename L::template split<index>;
00722
00723
                   using left = typename s::head;
                   using right = typename s::tail;
00724
00725
                   using ll = typename left::template push_back<T>;
00726
                   using type = typename ll::template concat<right>;
00727
00728
00729
              template <size_t index, typename L>
              struct remove_h {
00731
                   using s = typename L::template split<index>;
00732
                   using left = typename s::head;
                   using right = typename s::tail;
00733
00734
                   using rr = typename right::pop_front::tail;
00735
                   using type = typename left::template concat<rr>;
00736
00737
          } // namespace internal
00738
00741
          template <typename... Ts>
00742
          struct type_list {
00743
           private:
00744
              template <typename T>
00745
              struct concat_h;
00746
00747
              template <typename... Us>
00748
              struct concat_h<type_list<Us...» {</pre>
00749
                   using type = type_list<Ts..., Us...>;
00750
              };
00751
00752
00754
              static constexpr size_t length = sizeof...(Ts);
00755
00758
              \texttt{template} \; \texttt{<typename} \; \; \texttt{T>}
00759
              using push_front = type_list<T, Ts...>;
00760
00763
              template <size_t index>
00764
               using at = internal::type_at_t<index, Ts...>;
00765
00767
              struct pop_front {
    using type = typename internal::pop_front_h<Ts...>::head;
00769
                   using tail = typename internal::pop_front_h<Ts...>::tail;
00771
00772
              };
00773
00776
               template <typename T>
00777
              using push_back = type_list<Ts..., T>;
00778
00781
               template <typename U>
00782
              using concat = typename concat_h<U>::type;
00783
00786
               template <size_t index>
00787
               struct split {
00788
               private:
00789
                   using inner = internal::split_h<index, type_list<>, type_list<Ts...»;</pre>
00790
00791
00792
                   using head = typename inner::head;
00793
                   using tail = typename inner::tail;
00794
00795
00799
              template <typename T, size_t index>
              using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00801
00804
              template <size_t index>
00805
              using remove = typename internal::remove_h<index, type_list<Ts...»::type;</pre>
00806
          };
00807
00809
          template <>
00810
          struct type_list<> {
00811
              static constexpr size_t length = 0;
00812
00813
              template <typename T>
              using push_front = type_list<T>;
00814
00815
00816
               template <typename T>
00817
               using push_back = type_list<T>;
00818
00819
              template <typename U> \,
00820
              using concat = U:
```

```
00822
              // TODO(jewave): assert index == 0
              template <typename T, size_t index>
using insert = type_list<T>;
00823
00824
00825
00826 } // namespace aerobus
00828 // i16
00829 #ifdef WITH_CUDA_FP16
00830 // i16
00831 namespace aerobus {
         struct i16 {
00833
00834
              using inner_type = int16_t;
00837
              template<int16_t x>
00838
              struct val {
00840
                 using enclosing_type = i16;
                  static constexpr int16_t v = x;
00842
00843
00846
                  template<typename valueType>
00847
                  static constexpr INLINED DEVICE valueType get() {
00848
                      return internal::template int16_convert_helper<valueType, x>::value();
00849
00850
                  using is_zero_t = std::bool_constant<x == 0>;
00852
00853
00855
                  static std::string to_string() {
00856
                      return std::to_string(x);
00857
00858
              } ;
00859
00861
              using zero = val<0>;
00863
              using one = val<1>;
00865
              static constexpr bool is_field = false;
00867
              static constexpr bool is_euclidean_domain = true;
00870
              template<auto x>
              using inject_constant_t = val<static_cast<int16_t>(x)>;
00871
00872
              template<typename v>
00874
              using inject_ring_t = v;
00875
           private:
00876
              template<typename v1, typename v2>
00877
00878
              struct add {
00879
                  using type = val<v1::v + v2::v>;
00880
00881
00882
              template<typename v1, typename v2>
              struct sub {
00883
                  using type = val<v1::v - v2::v>;
00884
00885
00886
00887
              template<typename v1, typename v2>
00888
              struct mul {
00889
                 using type = val<v1::v* v2::v>;
00890
00891
00892
              template<typename v1, typename v2>
00893
00894
                  using type = val<v1::v / v2::v>;
00895
00896
              template<typename v1, typename v2>
00897
00898
              struct remainder {
00899
                 using type = val<v1::v % v2::v>;
00900
00901
00902
              template<typename v1, typename v2>
00903
              struct at {
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00904
00905
00906
00907
              template<typename v1, typename v2>
00908
              struct lt {
00909
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00910
00911
00912
              template<typename v1, typename v2>
00913
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00914
00915
00916
00917
              template<typename v1>
00918
              struct pos {
00919
                  using type = std::bool_constant<(v1::v > 0)>;
00920
00921
00922
           public:
```

```
template<typename v1, typename v2>
00928
              using add_t = typename add<v1, v2>::type;
00929
00934
              template<typename v1, typename v2>
00935
              using sub_t = typename sub<v1, v2>::type;
00936
00941
              template<typename v1, typename v2>
00942
              using mul_t = typename mul<v1, v2>::type;
00943
00948
              template<typename v1, typename v2>
00949
              using div_t = typename div<v1, v2>::type;
00950
00955
              template<typename v1, typename v2>
00956
              using mod_t = typename remainder<v1, v2>::type;
00957
00962
              template<typename v1, typename v2>
00963
              using gt_t = typename gt<v1, v2>::type;
00964
              template<typename v1, typename v2>
00970
              using lt_t = typename lt<v1, v2>::type;
00971
00976
              template<typename v1, typename v2>
00977
              using eq_t = typename eq<v1, v2>::type;
00978
00982
              template<typename v1, typename v2>
              static constexpr bool eq_v = eq_t<v1, v2>::value;
00984
00989
              template<typename v1, typename v2>
00990
              using gcd_t = gcd_t < i16, v1, v2>;
00991
00995
              template < typename v >
00996
              using pos_t = typename pos<v>::type;
00997
01001
              template<typename v>
01002
              static constexpr bool pos_v = pos_t<v>::value;
01003 };
01004 } // namespace aerobus
01005 #endif
01006
01007 // i32
01008 namespace aerobus {
01010
         struct i32 {
             using inner type = int32 t;
01011
              template<int32_t x>
01014
01015
              struct val {
                  using enclosing_type = i32;
01017
01019
                  static constexpr int32_t v = x;
01020
01023
                  template<typename valueType>
                  static constexpr DEVICE valueType get() {
01024
                      return static_cast<valueType>(x);
01026
01027
01029
                  using is_zero_t = std::bool_constant<x == 0>;
01030
01032
                  static std::string to string() {
                     return std::to_string(x);
01034
01035
              };
01036
01038
              using zero = val<0>;
              using one = val<1>;
01040
01042
              static constexpr bool is_field = false;
01044
              static constexpr bool is_euclidean_domain = true;
01047
              template<auto x>
01048
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01049
01050
              template<typename v>
01051
              using inject ring t = v:
01053
01054
              template<typename v1, typename v2>
01055
              struct add {
                 using type = val<v1::v + v2::v>;
01056
01057
01058
01059
              template<typename v1, typename v2>
01060
01061
                  using type = val<v1::v - v2::v>;
01062
01063
01064
              template<typename v1, typename v2>
01065
01066
                  using type = val<v1::v* v2::v>;
01067
01068
01069
              template<tvpename v1, tvpename v2>
```

```
01070
             struct div {
                 using type = val<v1::v / v2::v>;
01071
01072
              };
01073
              template<typename v1, typename v2>
01074
01075
              struct remainder {
01076
                 using type = val<v1::v % v2::v>;
01077
01078
01079
              template<typename v1, typename v2>
01080
              struct qt {
                using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01081
01082
01083
01084
              template<typename v1, typename v2>
01085
              struct lt {
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01086
01087
01088
01089
              template<typename v1, typename v2>
01090
              struct eq {
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01091
01092
              }:
01093
01094
              template<typename v1>
01095
              struct pos {
01096
                  using type = std::bool_constant<(v1::v > 0)>;
01097
01098
01099
           public:
01104
              template<typename v1, typename v2>
01105
              using add_t = typename add<v1, v2>::type;
01106
01111
              template<typename v1, typename v2>
01112
              using sub_t = typename sub<v1, v2>::type;
01113
              template<typename v1, typename v2>
01118
01119
              using mul_t = typename mul<v1, v2>::type;
01120
01125
              template<typename v1, typename v2>
01126
              using div_t = typename div<v1, v2>::type;
01127
01132
              template<typename v1, typename v2>
01133
              using mod_t = typename remainder<v1, v2>::type;
01134
01139
              template<typename v1, typename v2>
01140
              using gt_t = typename gt<v1, v2>::type;
01141
01146
              template<typename v1, typename v2>
              using lt_t = typename lt<v1, v2>::type;
01147
01148
01153
              template<typename v1, typename v2>
01154
              using eq_t = typename eq<v1, v2>::type;
01155
              template<typename v1, typename v2>
01159
01160
              static constexpr bool eq v = eq t<v1, v2>::value;
01166
              template<typename v1, typename v2>
01167
              using gcd_t = gcd_t<i32, v1, v2>;
01168
01172
              template<typename v>
01173
             using pos_t = typename pos<v>::type;
01174
01178
              template<typename v>
01179
              static constexpr bool pos_v = pos_t<v>::value;
01180
01181 } // namespace aerobus
01182
01183 // i64
01184 namespace aerobus {
01186
       struct i64 {
01188
             using inner_type = int64_t;
01191
              template<int64_t x>
01192
              struct val {
                 using inner_type = int32_t;
01194
01196
                 using enclosing_type = i64;
01198
                 static constexpr int64_t v = x;
01199
01202
                  template<typename valueType>
                 static constexpr INLINED DEVICE valueType get() {
01203
                      return static_cast<valueType>(x);
01204
01205
01206
01208
                  using is_zero_t = std::bool_constant<x == 0>;
01209
                  static std::string to string() {
01211
01212
                      return std::to string(x);
```

```
01213
                   }
01214
01215
01218
              template<auto x>
01219
              using inject constant t = val<static cast<int64 t>(x)>;
01220
01225
              template<typename v>
01226
              using inject_ring_t = v;
01227
              using zero = val<0>;
01229
              using one = val<1>;
01231
01233
              static constexpr bool is field = false;
01235
              static constexpr bool is_euclidean_domain = true;
01236
01237
           private:
01238
              template<typename v1, typename v2>
01239
              struct add {
                  using type = val<v1::v + v2::v>;
01240
01241
01242
01243
              template<typename v1, typename v2>
01244
              struct sub {
                 using type = val<v1::v - v2::v>;
01245
01246
01247
01248
              template<typename v1, typename v2>
01249
              struct mul
                 using type = val<v1::v* v2::v>;
01250
01251
01252
01253
              template<typename v1, typename v2>
01254
              struct div {
01255
                  using type = val<v1::v / v2::v>;
01256
              };
01257
01258
              template<typename v1, typename v2>
01259
              struct remainder {
01260
                  using type = val<v1::v% v2::v>;
01261
01262
01263
              template<typename v1, typename v2>
01264
              struct at {
                 using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01265
01266
01267
01268
              template<typename v1, typename v2>
01269
               struct lt {
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01270
01271
01272
              template<typename v1, typename v2>
01274
              struct eq {
01275
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01276
01277
01278
              template<typename v>
01279
              struct pos {
01280
                  using type = std::bool_constant<(v::v > 0)>;
01281
01282
01283
           public:
01287
              template<typename v1, typename v2>
01288
              using add_t = typename add<v1, v2>::type;
01289
01293
              template<typename v1, typename v2>
01294
              using sub_t = typename sub<v1, v2>::type;
01295
01299
              template<typename v1, typename v2>
01300
              using mul t = typename mul<v1, v2>::type;
01306
              template<typename v1, typename v2>
01307
              using div_t = typename div<v1, v2>::type;
01308
01312
              template<typename v1, typename v2>
              using mod_t = typename remainder<v1, v2>::type;
01313
01314
01319
               template<typename v1, typename v2>
01320
              using gt_t = typename gt<v1, v2>::type;
01321
              template<typename v1, typename v2> static constexpr bool gt_v = gt_t<v1, v2>::value;
01326
01327
01328
01333
               template<typename v1, typename v2>
01334
              using lt_t = typename lt<v1, v2>::type;
01335
              template<typename v1, typename v2>
static constexpr bool lt_v = lt_t<v1, v2>::value;
01340
01341
```

```
01342
01347
              template<typename v1, typename v2>
01348
              using eq_t = typename eq<v1, v2>::type;
01349
              template<typename v1, typename v2>
static constexpr bool eq_v = eq_t<v1, v2>::value;
01354
01355
01356
01361
              template<typename v1, typename v2>
01362
              using gcd_t = gcd_t < i64, v1, v2>;
01363
              template<typename v>
01367
01368
              using pos_t = typename pos<v>::type;
01369
01373
              template<typename v>
01374
              static constexpr bool pos_v = pos_t<v>::value;
01375
         };
01376
01378
          template<>
01379
          struct Embed<i32, i64> {
01382
              template<typename val>
01383
              using type = i64::val<static_cast<int64_t>(val::v)>;
01384
01385 } // namespace aerobus
01386
01387 // z/pz
01388 namespace aerobus {
01394
          template<int32_t p>
01395
          struct zpz {
              using inner_type = int32_t;
01397
01398
01401
              template<int32 t x>
01402
              struct val {
01404
                 using enclosing_type = zpz;
01406
                  static constexpr int32_t v = x % p;
01407
01410
                  template<typename valueType>
                  static constexpr INLINED DEVICE valueType get() {
01411
                      return static_cast<valueType>(x % p);
01412
01413
01414
01416
                  using is_zero_t = std::bool_constant<v == 0>;
01417
                  static constexpr bool is zero v = v == 0:
01419
01420
                  static std::string to_string() {
01423
01424
                      return std::to_string(x % p);
01425
01426
              };
01427
01430
              template<auto x>
01431
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01432
01434
              using zero = val<0>;
01435
              using one = val<1>:
01437
01438
              static constexpr bool is_field = is_prime::value;
01441
01443
              static constexpr bool is_euclidean_domain = true;
01444
01445
           private:
              template<typename v1, typename v2>
01446
01447
              struct add {
01448
                  using type = val<(v1::v + v2::v) % p>;
01449
              } ;
01450
01451
              template<typename v1, typename v2>
01452
              struct sub {
                  using type = val<(v1::v - v2::v) % p>;
01453
01454
01455
01456
              template<typename v1, typename v2>
01457
              struct mul {
                  using type = val<(v1::v* v2::v) % p>;
01458
01459
01460
01461
              template<typename v1, typename v2>
01462
              struct div {
                  using type = val<(v1::v% p) / (v2::v % p)>;
01463
01464
01465
01466
              template<typename v1, typename v2>
              struct remainder {
01467
01468
                  using type = val<(v1::v% v2::v) % p>;
01469
01470
01471
              template<tvpename v1, tvpename v2>
```

```
01472
              struct qt {
                  using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01473
01474
01475
01476
              template<typename v1, typename v2>
01477
              struct 1t {
01478
                  using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01479
01480
01481
              template<typename v1, typename v2>
01482
              struct eq {
                using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01483
01484
01485
01486
              template<typename v1>
01487
              struct pos {
01488
                  using type = std::bool_constant<(v1::v > 0)>;
01489
01490
           public:
01491
01495
              template<typename v1, typename v2>
01496
              using add_t = typename add<v1, v2>::type;
01497
01501
              template<typename v1, typename v2>
01502
              using sub_t = typename sub<v1, v2>::type;
01503
01507
              template<typename v1, typename v2>
01508
              using mul_t = typename mul<v1, v2>::type;
01509
01513
              template<typename v1, typename v2>
01514
              using div t = typename div<v1, v2>::type;
01515
01519
              template<typename v1, typename v2>
01520
              using mod_t = typename remainder<v1, v2>::type;
01521
01525
              template<typename v1, typename v2>
01526
              using gt_t = typename gt<v1, v2>::type;
01531
              template<typename v1, typename v2>
01532
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01533
01537
              template<typename v1, typename v2>
01538
              using lt_t = typename lt<v1, v2>::type;
01539
              template<typename v1, typename v2> static constexpr bool lt_v = lt_t < v1, v2>::value;
01543
01544
01545
01549
              template<typename v1, typename v2>
              using eq_t = typename eq<v1, v2>::type;
01550
01551
              template<typename v1, typename v2>
01556
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01557
01561
              template<typename v1, typename v2>
              using gcd_t = gcd_t < i32, v1, v2>;
01562
01563
01566
              template<typename v1>
01567
              using pos_t = typename pos<v1>::type;
01568
01571
              template < typename v >
              static constexpr bool pos_v = pos_t<v>::value;
01572
01573
         };
01577
          template<int32_t x>
01578
          struct Embed<zpz<x>, i32> {
01581
              template <typename val>
01582
              using type = i32::val<val::v>;
01583
          };
01584 } // namespace aerobus
01586 // polynomial
01587 namespace aerobus {
01588
          // coeffN x^N + ...
01593
          template<typename Ring>
01594
          requires IsEuclideanDomain<Ring>
01595
          struct polynomial {
01596
             static constexpr bool is_field = false;
01597
              static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01598
01601
              template<tvpename P>
              struct horner_reduction_t {
01602
01603
                  template<size_t index, size_t stop>
01604
                  struct inner {
01605
                      template<typename accum, typename x>
01606
                      using type = typename horner_reduction_t<P>::template inner<index + 1, stop>
01607
                           ::template type<
01608
                              typename Ring::template add t<
```

```
typename Ring::template mul_t<x, accum>,
                                   typename P::template coeff_at_t<P::degree - index>
01610
                              >, x>;
01611
01612
                  };
01613
01614
                  template<size t stop>
                  struct inner<stop, stop> {
01615
                      template<typename accum, typename x>
01616
01617
                      using type = accum;
01618
                  } ;
              };
01619
01620
              template<typename coeffN, typename... coeffs>
01624
01625
01627
                  using ring_type = Ring;
01629
                  using enclosing_type = polynomial<Ring>;
01631
                  static constexpr size_t degree = sizeof...(coeffs);
                  using aN = coeffN;
01633
01635
                  using strip = val<coeffs...>;
                  using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01637
01639
                  static constexpr bool is_zero_v = is_zero_t::value;
01640
01641
               private:
                  template<size_t index, typename E = void>
01642
01643
                  struct coeff_at {};
01644
01645
                  template<size_t index>
01646
                  struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))» {</pre>
01647
                      using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01648
01649
01650
                  template<size_t index>
01651
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))» {
01652
                      using type = typename Ring::zero;
01653
01654
01655
               public:
01658
                  template<size_t index>
01659
                  using coeff_at_t = typename coeff_at<index>::type;
01660
01663
                  static std::string to_string() {
                      return string_helper<coeffN, coeffs...>::func();
01664
01665
01666
01671
                  template<typename arithmeticType>
01672
                  static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01673
                      #ifdef WITH CUDA FP16
01674
                      arithmeticType start;
                      if constexpr (std::is_same_v<arithmeticType, __half2>) {
01675
01676
                          start = \underline{\quad }half2(0, 0);
01677
                      } else {
01678
                          start = static_cast<arithmeticType>(0);
01679
01680
                      #6156
                      arithmeticType start = static_cast<arithmeticType>(0);
01681
01682
                      #endif
01683
                      return horner_evaluation<arithmeticType, val>
01684
                              ::template inner<0, degree + 1>
01685
                              ::func(start, x);
01686
                  }
01687
01700
                  template<typename arithmeticType>
01701
                  static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01702
                     return compensated_horner<arithmeticType, val>::func(x);
01703
                  }
01704
01705
                  template<typename x>
                  using value_at_t = horner_reduction_t<val>
01706
01707
                      ::template inner<0, degree + 1>
01708
                      ::template type<typename Ring::zero, x>;
01709
              };
01710
01713
              template<typename coeffN>
              struct val<coeffN> {
01714
01716
                  using ring_type = Ring;
01718
                  using enclosing_type = polynomial<Ring>;
01720
                  static constexpr size_t degree = 0;
01721
                  using aN = coeffN;
                  using strip = val<coeffN>;
01722
                  using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01723
01724
                  static constexpr bool is_zero_v = is_zero_t::value;
01726
01727
                  template<size_t index, typename E = void>
01728
                  struct coeff_at {};
01729
01730
                  template<size t index>
```

```
struct coeff_at<index, std::enable_if_t<(index == 0)» {</pre>
01732
                      using type = aN;
01733
                  };
01734
01735
                  template<size t index>
01736
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01737
                      using type = typename Ring::zero;
01738
01739
01740
                  template<size_t index>
01741
                  using coeff_at_t = typename coeff_at<index>::type;
01742
01743
                  static std::string to_string() {
01744
                      return string_helper<coeffN>::func();
01745
01746
01747
                  template<typename arithmeticType>
01748
                  static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01749
                      return coeffN::template get<arithmeticType>();
01750
                  }
01751
01752
                  template<typename arithmeticType>
                  static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01753
01754
                      return coeffN::template get<arithmeticType>();
01755
01756
                  template<typename x>
01757
01758
                  using value_at_t = coeffN;
01759
              };
01760
01762
              using zero = val<typename Ring::zero>;
01764
              using one = val<typename Ring::one>;
01766
              using X = val<typename Ring::one, typename Ring::zero>;
01767
           private:
01768
              template<typename P, typename E = void>
01769
01770
              struct simplify;
01771
01772
              template <typename P1, typename P2, typename I>
01773
01774
01775
              template<typename P1, typename P2>
01776
              struct add {
01777
                  using type = typename simplify<typename add_low<
01778
01779
                  P2.
01780
                  internal::make_index_sequence_reverse<</pre>
01781
                  std::max(P1::degree, P2::degree) + 1
01782
                  »::type>::type;
01783
01784
01785
              template <typename P1, typename P2, typename I>
01786
              struct sub_low;
01787
01788
              template <typename P1, typename P2, typename I>
01789
              struct mul low;
01790
01791
              template<typename v1, typename v2>
01792
              struct mul {
01793
                      using type = typename mul_low<
01794
                          v1,
01795
                          v2,
01796
                          internal::make_index_sequence_reverse<
01797
                          v1::degree + v2::degree + 1
01798
                          »::type;
01799
01800
              template<typename coeff, size_t deg>
01801
01802
              struct monomial:
01804
              template<typename v, typename E = void>
01805
              struct derive_helper {};
01806
01807
              template<typename v>
01808
              struct derive helper<v, std::enable if t<v::degree == 0» {
01809
                  using type = zero;
01810
01811
01812
              template<typename v>
              struct derive_helper<v, std::enable_if_t<v::degree != 0» {</pre>
01813
01814
                  using type = typename add<
01815
                      typename derive_helper<typename simplify<typename v::strip>::type>::type,
01816
                      typename monomial<
01817
                          typename Ring::template mul_t<
01818
                              typename v::aN,
01819
                               typename Ring::template inject_constant_t<(v::degree)>
01820
```

```
v::degree - 1
                        >::type
01822
01823
                    >::type;
01824
               };
01825
01826
                template<typename v1, typename v2, typename E = void>
01827
                struct eq_helper {};
01828
01829
                template<typename v1, typename v2>
               struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
   using type = std::false_type;</pre>
01830
01831
01832
01833
01834
01835
                template<typename v1, typename v2>
                struct eq_helper<v1, v2, std::enable_if_t<
    v1::degree == v2::degree &&</pre>
01836
01837
                    (v1::degree != 0 || v2::degree != 0) &&
01838
01839
                    std::is_same<
01840
                    typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
                    std::false_type
01841
01842
                    >::value
01843
               >
               > {
01844
01845
                    using type = std::false_type;
01846
                };
01847
                template<typename v1, typename v2>
struct eq_helper<v1, v2, std::enable_if_t<
    v1::degree == v2::degree &&</pre>
01848
01849
01850
01851
                    (v1::degree != 0 || v2::degree != 0) &&
01852
                    std::is_same<
01853
                    typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01854
                    std::true_type
                    >::value
01855
01856
                » {
01857
                    using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
                };
01859
01860
                template<typename v1, typename v2>
01861
                struct eq_helper<v1, v2, std::enable_if_t<
                    v1::degree == v2::degree &&
01862
                    (v1::degree == 0)
01863
01864
                » {
01865
                    using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01866
01867
01868
                template<typename v1, typename v2, typename E = void>
01869
                struct lt_helper {};
01870
01871
                template<typename v1, typename v2>
01872
               struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
01873
                    using type = std::true_type;
01874
01875
01876
                template<typename v1, typename v2>
01877
                struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01878
                    using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01879
01880
               template<typename v1, typename v2>
struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
01881
01882
01883
                    using type = std::false_type;
01884
01885
01886
                template<typename v1, typename v2, typename E = void>
01887
                struct gt_helper {};
01888
                template<typename v1, typename v2>
struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
01889
01891
                   using type = std::true_type;
01892
01893
                template<typename v1, typename v2>
struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01894
01895
01896
                    using type = std::false_type;
01897
01898
                template<typename v1, typename v2>
struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {</pre>
01899
01900
                    using type = std::false_type;
01901
01902
01903
01904
                // when high power is zero : strip
01905
                template<typename P>
01906
                struct simplify<P, std::enable_if_t<
01907
                    std::is same<
```

```
typename Ring::zero,
01909
                   typename P::aN
01910
                   >::value && (P::degree > 0)
01911
01912
                  using type = typename simplify<typename P::strip>::type;
01913
              };
01914
01915
               // otherwise : do nothing
01916
               template<typename P>
01917
              struct simplify<P, std::enable_if_t<
01918
                  !std::is_same<
01919
                  typename Ring::zero.
01920
                  typename P::aN
01921
                  >::value && (P::degree > 0)
01922
01923
                   using type = P;
01924
              };
01925
01926
              // do not simplify constants
01927
              template<typename P>
01928
              struct simplify<P, std::enable_if_t<P::degree == 0» {</pre>
01929
                  using type = P;
01930
              };
01931
01932
              // addition at
01933
               template<typename P1, typename P2, size_t index>
01934
              struct add_at {
01935
                  using type =
01936
                       typename Ring::template add_t<</pre>
                           typename P1::template coeff_at_t<index>,
01937
01938
                           typename P2::template coeff_at_t<index»;</pre>
01939
              };
01940
01941
               template<typename P1, typename P2, size_t index>
01942
              using add_at_t = typename add_at<P1, P2, index>::type;
01943
              template<typename P1, typename P2, std::size_t... I>
struct add_low<P1, P2, std::index_sequence<I...» {</pre>
01944
01945
01946
                  using type = val<add_at_t<P1, P2, I>...>;
01947
01948
              // substraction at
01949
01950
              template<typename P1, typename P2, size_t index>
01951
              struct sub_at {
01952
                  using type =
01953
                       typename Ring::template sub_t<
01954
                           typename P1::template coeff_at_t<index>,
01955
                           typename P2::template coeff_at_t<index»;</pre>
01956
              };
01957
01958
               template<typename P1, typename P2, size_t index>
01959
              using sub_at_t = typename sub_at<P1, P2, index>::type;
01960
01961
              template<typename P1, typename P2, std::size_t... I>
01962
              struct sub_low<P1, P2, std::index_sequence<I...» {
                  using type = val<sub_at_t<P1, P2, I>...>;
01963
01964
01965
01966
               template<typename P1, typename P2>
01967
               struct sub {
01968
                  using type = typename simplify<typename sub_low<
01969
                  P1,
01970
                  P2,
01971
                  internal::make_index_sequence_reverse<
01972
                   std::max(P1::degree, P2::degree) + 1
01973
                   »::type>::type;
01974
01975
01976
               // multiplication at
               template<typename v1, typename v2, size_t k, size_t index, size_t stop>
01978
               struct mul_at_loop_helper {
01979
                   using type = typename Ring::template add_t<
                       typename Ring::template mul_t<</pre>
01980
                       typename v1::template coeff_at_t<index>,
01981
                       typename v2::template coeff_at_t<k - index>
01982
01983
01984
                       typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
01985
01986
              };
01987
01988
              template<typename v1, typename v2, size_t k, size_t stop>
01989
              struct mul_at_loop_helper<v1, v2, k, stop, stop> {
01990
                   using type = typename Ring::template mul_t<
01991
                       typename v1::template coeff_at_t<stop>,
01992
                       typename v2::template coeff_at_t<0»;</pre>
01993
              };
01994
```

```
template <typename v1, typename v2, size_t k, typename E = void>
01996
01997
               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)» {
    using type = typename Ring::zero;
01998
01999
02000
02001
02002
02003
                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)» {
02004
02005
                    using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
02006
02007
02008
                template<typename P1, typename P2, size_t index>
02009
                using mul_at_t = typename mul_at<P1, P2, index>::type;
02010
               template<typename P1, typename P2, std::size_t... I>
struct mul_low<P1, P2, std::index_sequence<I...» {
    using type = val<mul_at_t<P1, P2, I>...>;
02011
02012
02013
02014
               };
02015
02016
               // division helper
02017
               template< typename A, typename B, typename Q, typename R, typename E = void>
02018
               struct div helper {};
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)» {
02024
                    using q_type = Q;
02025
                    using mod_type = R;
02026
                    using gcd type = B;
02027
02028
02029
               template<typename A, typename B, typename Q, typename R>
               struct div_helper<A, B, Q, R, std::enable_if_t<
(R::degree >= B::degree) &&
02030
02031
                    !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
02033
                private: // NOLINT
                   using rN = typename R::aN;
02034
02035
                    using bN = typename B::aN;
                    using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
02036
     B::degree>::type;
02037
                    using rr = typename sub<R, typename mul<pT, B>::type>::type;
                    using qq = typename add<Q, pT>::type;
02038
02039
                public:
02040
                   using q_type = typename div_helper<A, B, qq, rr>::q_type;
02041
02042
                    using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
                    using gcd_type = rr;
02043
02044
               };
02045
02046
               template<typename A, typename B>
02047
               struct div {
                    static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
02048
02049
                    using q_type = typename div_helper<A, B, zero, A>::q_type; using m_type = typename div_helper<A, B, zero, A>::mod_type;
02050
02051
               };
02052
02053
               template<typename P>
02054
               struct make unit {
02055
                    using type = typename div<P, val<typename P::aN»::q_type;
02056
02057
02058
               template<typename coeff, size_t deg>
02059
               struct monomial {
02060
                    using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
02061
02062
02063
               template<typename coeff>
02064
               struct monomial < coeff, 0 > {
02065
                    using type = val<coeff>;
02066
02067
02068
               template<typename arithmeticType, typename P>
               struct horner_evaluation {
02069
02070
                    template<size_t index, size_t stop>
02071
                    struct inner {
                         static constexpr DEVICE INLINED arithmeticType func(
02072
                             const arithmeticType& accum, const arithmeticType& x) {
  return horner_evaluation<arithmeticType, P>::template inner<index + 1,</pre>
02073
02074
      stop>::func(
02075
                                  internal::fma_helper<arithmeticType>::eval(
02076
02077
                                      accum,
                                      P::template coeff_at_t<P::degree - index>::template
02078
      get<arithmeticTvpe>()), x);
```

```
02080
02081
02082
                   template<size_t stop>
02083
                   struct inner<stop, stop> {
                       static constexpr DEVICE INLINED arithmeticType func(
02084
                           const arithmeticType& accum, const arithmeticType& x) {
02086
                            return accum;
02087
02088
                   };
02089
              };
02090
02091
               template<typename arithmeticType, typename P>
02092
               struct compensated_horner {
02093
                   template<int64_t index, int ghost>
                   struct EFTHorner {
   static INLINED void func(
02094
02095
02096
                                arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType
      *r) {
02097
                            arithmeticType p;
                            internal::two_prod(*r, x, &p, pi + P::degree - index - 1);
constexpr arithmeticType coeff = P::template coeff_at_t<index>::template
02098
02099
      get<arithmeticType>();
02100
                            internal::two_sum<arithmeticType>(
                                p, coeff,
r, sigma + P::degree - index - 1);
02101
02102
02103
                            EFTHorner<index - 1, ghost>::func(x, pi, sigma, r);
02104
02105
                   };
02106
02107
                   template<int ghost>
02108
                   struct EFTHorner<-1, ghost> {
02109
                      static INLINED DEVICE void func(
02110
                                \verb|arithmeticType| x, arithmeticType| *pi, arithmeticType| *sigma, arithmeticType|
      *r) {
02111
02112
                   };
02113
02114
                   static INLINED DEVICE arithmeticType func(arithmeticType x) {
02115
                       arithmeticType pi[P::degree], sigma[P::degree];
                       arithmeticType r = P::template coeff_at_t<P::degree>::template get<arithmeticType>();
EFTHorner<P::degree - 1, 0>::func(x, pi, sigma, &r);
02116
02117
                       arithmeticType c = internal::horner<arithmeticType, P::degree - 1>(pi, sigma, x);
02118
02119
                        return r + c;
02120
                   }
02121
               };
02122
02123
               template<typename coeff, typename... coeffs>
02124
               struct string_helper {
02125
                   static std::string func() {
02126
                        std::string tail = string_helper<coeffs...>::func();
02127
                        std::string result = "";
02128
                        if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
                        return tail;
} else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
02129
02130
02131
                            if (sizeof...(coeffs) == 1) {
02132
                                result += "x";
02133
                            } else {
02134
                                result += "x^" + std::to_string(sizeof...(coeffs));
02135
02136
                        } else {
02137
                            if (sizeof...(coeffs) == 1) {
02138
                                result += coeff::to_string() + " x";
02139
02140
                                 result += coeff::to_string()
02141
                                         + " x^* + std::to_string(sizeof...(coeffs));
02142
02143
                        }
02144
02145
                        if (!tail.empty()) {
                            if (tail.at(0) != '-') {
    result += " + " + tail;
02146
02147
02148
                            } else {
                                result += " - " + tail.substr(1);
02149
02150
02151
02152
02153
                        return result;
02154
                   }
02155
              }:
02156
02157
               template<typename coeff>
02158
               struct string_helper<coeff> {
02159
                   static std::string func() {
02160
                       if (!std::is_same<coeff, typename Ring::zero>::value) {
02161
                            return coeff::to_string();
                        } else {
02162
```

```
02163
                          return "";
02164
02165
                 }
02166
             };
02167
02168
          public:
              template<typename P>
02171
02172
              using simplify_t = typename simplify<P>::type;
02173
02177
              template<typename v1, typename v2>
02178
             using add_t = typename add<v1, v2>::type;
02179
02183
              template<typename v1, typename v2>
02184
              using sub_t = typename sub<v1, v2>::type;
02185
02189
              template<typename v1, typename v2>
02190
              using mul_t = typename mul<v1, v2>::type;
02191
02195
              template<typename v1, typename v2>
02196
              using eq_t = typename eq_helper<v1, v2>::type;
02197
02201
              template<typename v1, typename v2>
02202
              using lt_t = typename lt_helper<v1, v2>::type;
02203
02207
              template<typename v1, typename v2>
02208
              using gt_t = typename gt_helper<v1, v2>::type;
02209
02213
              template<typename v1, typename v2>
02214
              using div_t = typename div<v1, v2>::q_type;
02215
02219
              template<typename v1, typename v2>
02220
              using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
02221
02225
              template<typename coeff, size_t deg>
02226
              using monomial_t = typename monomial<coeff, deg>::type;
02227
02230
              template<typename v>
02231
              using derive_t = typename derive_helper<v>::type;
02232
02235
              template<typename v>
02236
              using pos_t = typename Ring::template pos_t<typename v::aN>;
02237
02240
              template<tvpename v>
02241
              static constexpr bool pos_v = pos_t<v>::value;
02242
02246
              template<typename v1, typename v2>
02247
              using gcd_t = std::conditional_t<</pre>
02248
                  Ring::is_euclidean_domain,
                  typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
02249
02250
                  void>;
02251
02254
              template<auto x>
02255
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
02256
02259
              template<typename v>
02260
              using inject_ring_t = val<v>;
02261
          };
02262 } // namespace aerobus
02263
02264 // fraction field
02265 namespace aerobus {
02266
         namespace internal {
02267
             template<typename Ring, typename E = void>
02268
              requires IsEuclideanDomain<Ring>
02269
             struct _FractionField {};
02270
02271
             template<typename Ring>
02272
              requires IsEuclideanDomain<Ring>
02273
              struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
                  static constexpr bool is_field = true;
02276
                  static constexpr bool is_euclidean_domain = true;
02277
              private:
02278
                  template<typename val1, typename val2, typename E = void>
02279
02280
                  struct to_string_helper {};
02281
02282
                  template<typename val1, typename val2>
02283
                  struct to_string_helper <val1, val2,
02284
                      std::enable_if_t<
02285
                      Ring::template eg t<
02286
                      val2, typename Ring::one
02287
                      >::value
02288
02289
02290
                      static std::string func() {
02291
                          return vall::to_string();
02292
```

```
};
02293
02294
02295
                  template<typename val1, typename val2>
02296
                  struct to_string_helper<val1, val2,
02297
                      std::enable if t<
02298
                       !Ring::template eq_t<
02299
                      val2,
02300
                      typename Ring::one
02301
                      >::value
02302
02303
                  > {
02304
                      static std::string func() {
02305
                          return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
02306
02307
                  };
02308
               public:
02309
02313
                  template<typename val1, typename val2>
02314
                  struct val {
02316
                      using x = val1;
                      using y = val2;
02318
02320
                      using is_zero_t = typename val1::is_zero_t;
02322
                      static constexpr bool is_zero_v = val1::is_zero_t::value;
02323
02325
                      using ring_type = Ring;
                      using enclosing_type = _FractionField<Ring>;
02326
02327
02330
                       static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
02331
02335
                      template<typename valueType>
                      static constexpr INLINED DEVICE valueType get() {
02336
02337
                          return internal::staticcast<valueType, typename ring_type::inner_type>::template
      func<x::v>() /
02338
                               internal::staticcast<valueType, typename ring_type::inner_type>::template
      func<y::v>();
02339
02340
02343
                      static std::string to_string() {
02344
                          return to_string_helper<val1, val2>::func();
02345
02346
02351
                      template<typename arithmeticType>
                      static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& v) {
02352
02353
                          return x::eval(v) / y::eval(v);
02354
02355
                  } ;
02356
02358
                  using zero = val<typename Ring::zero, typename Ring::one>;
02360
                  using one = val<typename Ring::one, typename Ring::one>;
02361
02364
                  template<typename v>
02365
                  using inject_t = val<v, typename Ring::one>;
02366
02369
                  template<auto x>
02370
                  using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
     Ring::one>;
02371
02374
                  template<typename v>
02375
                  using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
02376
02378
                  using ring_type = Ring;
02379
02380
               private:
02381
                  template<typename v, typename E = void>
02382
                  struct simplify {};
02383
02384
02385
                  template<tvpename v>
02386
                  struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {
                      using type = typename _FractionField<Ring>::zero;
02387
02388
02389
02390
                  // x != 0
02391
                  template<typename v>
02392
                  struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value» {</pre>
02393
02394
                      using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
02395
                      using newx = typename Ring::template div_t<typename v::x, _gcd>;
02396
                      using newy = typename Ring::template div_t<typename v::y, _gcd>;
02397
02398
                      using posx = std::conditional t<
02399
                                           !Ring::template pos_v<newy>,
02400
                                           typename Ring::template sub_t<typename Ring::zero, newx>,
                                           newx>;
02401
02402
                      using posy = std::conditional_t<
02403
                                           !Ring::template pos_v<newy>,
02404
                                           typename Ring::template sub t<typename Ring::zero, newy>,
```

```
02405
                                           newy>;
02406
                   public:
02407
                      using type = typename _FractionField<Ring>::template val<posx, posy>;
02408
                  };
02409
02410
               public:
02413
                  template<typename v>
02414
                  using simplify_t = typename simplify<v>::type;
02415
02416
02417
                  template<typename v1, typename v2>
02418
                  struct add {
02419
                   private:
02420
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02421
                      using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
                      using dividend = typename Ring::template add_t<a, b>;
using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02422
02423
02424
                      using g = typename Ring::template gcd_t<dividend, diviser>;
02425
02426
                   public:
                      using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
02427
     diviser»;
02428
                  };
02429
02430
                  template<typename v>
02431
                  struct pos {
02432
                      using type = std::conditional_t<
02433
                           (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>),
02434
02435
                          std::true type,
02436
                          std::false type>;
02437
                  };
02438
02439
                  template<typename v1, typename v2>
02440
                  struct sub {
                   private:
02441
02442
                      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>;
02443
02444
                      using dividend = typename Ring::template sub_t<a, b>;
02445
                      using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02446
                      using g = typename Ring::template gcd_t<dividend, diviser>;
02447
                   public:
02448
02449
                      using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
     diviser»;
02450
02451
02452
                  template<typename v1, typename v2>
02453
                  struct mul {
02454
                   private:
02455
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
02456
                      using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02457
                   public:
02458
                      using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
02459
02460
                  };
02461
02462
                  template<typename v1, typename v2, typename E = void>
02463
                  struct div {};
02464
02465
                  template<typename v1, typename v2>
                  struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
02466
      _FractionField<Ring>::zero>::value»
02467
                  private:
02468
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02469
                      using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02470
02471
                   public:
02472
                      using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
02473
02474
02475
                  template<typename v1, typename v2>
02476
                  struct div<v1, v2, std::enable_if_t<
02477
                      std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
02478
                      using type = one;
02479
02480
02481
                  template<typename v1, typename v2>
                  struct eq {
02482
02483
                      using type = std::conditional t<
                              std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
02484
02485
                               std::is_same<typename simplify_t<vl>::y, typename simplify_t<v2>::y>::value,
02486
                           std::true_type,
02487
                           std::false_type>;
02488
                  };
02489
02490
                  template<typename v1, typename v2, typename E = void>
```

```
02491
                   struct gt;
02492
02493
                   template<typename v1, typename v2>
02494
                   struct gt<v1, v2, std::enable_if_t<
                       (eq<v1, v2>::type::value)
02495
02496
                       using type = std::false_type;
02498
02499
02500
                   template<typename v1, typename v2>
                   struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02501
02502
02503
                       (!pos<v1>::type::value) && (!pos<v2>::type::value)
02504
02505
                       using type = typename gt<
02506
                           typename sub<zero, v1>::type, typename sub<zero, v2>::type
02507
                       >::type;
02508
                   };
02510
                   template<typename v1, typename v2>
                   struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02511
02512
02513
                       (pos<v1>::type::value) && (!pos<v2>::type::value)
02514
                       » {
02515
                       using type = std::true_type;
02516
                   };
02517
02518
                   template<typename v1, typename v2>
02519
                   struct gt<v1, v2, std::enable_if_t<
02520
                       (!eq<v1, v2>::type::value) &&
02521
                       (!pos<v1>::type::value) && (pos<v2>::type::value)
02522
02523
                       using type = std::false_type;
02524
                   } ;
02525
                   template<typename v1, typename v2>
02526
                   struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02527
02529
                       (pos<v1>::type::value) && (pos<v2>::type::value)
02530
02531
                       using type = typename Ring::template gt_t<
                           typename Ring::template mul_t<v1::x, v2::y>,
02532
02533
                           typename Ring::template mul_t<v2::y, v2::x>
02534
02535
                   };
02536
02537
                public:
02541
                   template<typename v1, typename v2>
                   using add_t = typename add<v1, v2>::type;
02542
02543
                   template<typename v1, typename v2>
02549
                   using mod_t = zero;
02550
02555
                   template<typename v1, typename v2>
02556
                   using gcd_t = v1;
02557
02561
                   template<typename v1, typename v2>
02562
                   using sub_t = typename sub<v1, v2>::type;
02563
02567
                   template<typename v1, typename v2>
02568
                   using mul_t = typename mul<v1, v2>::type;
02569
                   template<typename v1, typename v2>
02574
                   using div_t = typename div<v1, v2>::type;
02575
02579
                   template<typename v1, typename v2>
02580
                   using eq_t = typename eq<v1, v2>::type;
02581
02585
                   template<typename v1, typename v2>
                   static constexpr bool eq_v = eq<v1, v2>::type::value;
02587
02591
                   template<typename v1, typename v2>
02592
                   using gt_t = typename gt<v1, v2>::type;
02593
02597
                   template<typename v1, typename v2>
02598
                   static constexpr bool gt_v = gt<v1, v2>::type::value;
02599
02602
                   template<typename v1>
02603
                   using pos_t = typename pos<v1>::type;
02604
02607
                   template<typename v>
02608
                   static constexpr bool pos_v = pos_t < v > :: value;
02609
02610
02611
               template<typename Ring, typename E = void>
02612
               requires IsEuclideanDomain<Ring>
02613
               struct FractionFieldImpl {};
```

```
// fraction field of a field is the field itself
02615
02616
              template<typename Field>
              requires IsEuclideanDomain<Field>
02617
02618
              struct FractionFieldImpl<Field, std::enable if t<Field::is field» {
02619
                  using type = Field:
02620
                  template<typename v>
02621
                  using inject_t = v;
02622
             } ;
02623
              // fraction field of a ring is the actual fraction field
02624
02625
              template<tvpename Ring>
02626
              requires IsEuclideanDomain<Ring>
02627
              struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {</pre>
02628
                 using type = _FractionField<Ring>;
02629
         } // namespace internal
02630
02631
02634
          template<typename Ring>
02635
          requires IsEuclideanDomain<Ring>
02636
          using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02637
02640
          template<typename Ring>
          struct Embed<Ring, FractionField<Ring» {</pre>
02641
02644
              template<typename v>
02645
              using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02646
02647 } // namespace aerobus
02648
02649
02650 // short names for common types
02651 namespace aerobus {
02655
       template<typename X, typename Y>
02656
          requires IsRing<typename X::enclosing_type> &&
02657
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
          using add_t = typename X::enclosing_type::template add_t<X, Y>;
02658
02659
02663
          template<typename X, typename Y>
02664
          requires IsRing<typename X::enclosing_type> &&
02665
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02666
          using sub_t = typename X::enclosing_type::template sub_t<X, Y>;
02667
02671
          template<typename X, typename Y>
02672
          requires IsRing<typename X::enclosing_type> &&
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02673
02674
          using mul_t = typename X::enclosing_type::template mul_t<X, Y>;
02675
02679
          template<typename X, typename Y>
          requires IsEuclideanDomain<typename X::enclosing_type> &&
02680
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02681
          using div_t = typename X::enclosing_type::template div_t<X, Y>;
02682
02683
02686
          using q32 = FractionField<i32>;
02687
02690
          using fpq32 = FractionField<polynomial<q32>>;
02691
02694
          using g64 = FractionField<i64>;
02695
02697
          using pi64 = polynomial<i64>;
02698
02700
          using pg64 = polynomial<g64>;
02701
02703
          using fpq64 = FractionField<polynomial<q64>>;
02704
02709
          template<typename Ring, typename v1, typename v2>
02710
          using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02711
02718
          template<typename v>
02719
          using embed_int_poly_in_fractions_t =
02720
                  typename Embed<
02721
                      polynomial<typename v::ring_type>,
02722
                      polynomial<FractionField<typename v::ring_type>»::template type<v>;
02723
02727
          template<int64_t p, int64_t q>
02728
          using make_q64_t = typename q64::template simplify_t<
02729
                      typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02730
02734
          template<int32_t p, int32_t q>
          using make_q32_t = typename q32::template simplify_t<
02735
02736
                      typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>»;
02737
02742
          template<typename Ring, typename v1, typename v2>
02743
          using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02748
          template<typename Ring, typename v1, typename v2>
02749
          using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02750
02752
          template<>
```

```
struct Embed<q32, q64> {
             template<typename v>
02756
02757
              using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02758
          };
02759
02763
          template<typename Small, typename Large>
02764
         struct Embed<polynomial<Small>, polynomial<Large» {</pre>
02765
02766
             template<typename v, typename i>
02767
              struct at_low;
02768
02769
             template<typename v, size_t i>
02770
             struct at index {
                 using type = typename Embed<Small, Large>::template
02771
     type<typename v::template coeff_at_t<i>>;
02772
02773
02774
              template<typename v, size t... Is>
02775
             struct at_low<v, std::index_sequence<Is...» {
02776
                using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
02777
02778
02779
          public:
02782
             template<typename v>
02783
             using type = typename at_low<v, typename internal::make_index_sequence_reverse<v::degree +
     1»::type;
02784
         };
02785
02789
          template<typename Ring, auto... xs>
02790
         using make_int_polynomial_t = typename polynomial<Ring>::template val<</pre>
02791
                  typename Ring::template inject_constant_t<xs>...>;
02792
02796
         template<typename Ring, auto... xs>
02797
         using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<</pre>
02798
                 typename FractionField<Ring>::template inject_constant_t<xs>...>;
02799 } // namespace aerobus
02800
02801 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02802 namespace aerobus {
02803
       namespace internal {
02804
             template<typename T, size_t x, typename E = void>
02805
             struct factorial { };
02806
02807
             template<typename T, size_t x>
02808
              struct factorial<T, x, std::enable_if_t<(x > 0)  {
02809
             private:
02810
                  template<typename, size_t, typename>
02811
                  friend struct factorial;
              public:
02812
                 using type = typename T::template mul t<typename T::template val<x>, typename factorial<T,
02813
     x - 1>::type>;
02814
                 static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02815
            };
02816
02817
             template<typename T>
02818
             struct factorial<T, 0> {
02819
              public:
02820
                 using type = typename T::one;
02821
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02822
              };
02823
          } // namespace internal
02824
02828
          template<typename T, size_t i>
02829
          using factorial_t = typename internal::factorial<T, i>::type;
02830
02834
          template<typename T, size_t i>
02835
         inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02837
          namespace internal {
02838
              template<typename T, size_t k, size_t n, typename E = void>
02839
             struct combination_helper {};
02840
02841
              template<typename T, size t k, size t n>
02842
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
02843
                 using type = typename FractionField<T>::template mul_t<</pre>
02844
                      typename combination_helper<T, k - 1, n - 1>::type,
02845
                      makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02846
             }:
02847
02848
              template<typename T, size_t k, size_t n>
02849
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)   {
02850
                  using type = typename combination_helper<T, n - k, n>::type;
02851
02852
02853
              template<typename T, size t n>
```

```
struct combination_helper<T, 0, n> {
                  using type = typename FractionField<T>::one;
02855
02856
               };
02857
02858
               template<typename T, size_t k, size_t n>
02859
               struct combination {
02860
                   using type = typename internal::combination_helper<T, k, n>::type::x;
02861
                   static constexpr typename T::inner_type value
02862
                               internal::combination_helper<T, k, n>::type::template get<typename</pre>
      T::inner_type>();
02863
               };
02864
          } // namespace internal
02865
02868
          template<typename T, size_t k, size_t n>
02869
          using combination_t = typename internal::combination<T, k, n>::type;
02870
          template<typename T, size_t k, size_t n>
inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02875
02876
02877
02878
          namespace internal {
02879
               template<typename T, size_t m>
02880
               struct bernoulli;
02881
               template<typename T, typename accum, size_t k, size_t m>
struct bernoulli_helper {
02882
02883
                  using type = typename bernoulli_helper<
02884
02885
02886
                       addfractions_t<T,
02887
                           accum,
                            mulfractions_t<T,</pre>
02888
02889
                                makefraction t<T.
02890
                                    combination_t<T, k, m + 1>,
02891
                                    typename T::one>,
02892
                                typename bernoulli<T, k>::type
02893
                       >,
k + 1,
02894
02895
02896
                       m>::type;
02897
               };
02898
02899
               template<typename T, typename accum, size_t m>
               struct bernoulli_helper<T, accum, m, m> {
02900
                  using type = accum;
02901
02902
02903
02904
02905
02906
               template<typename T, size_t m>
02907
               struct bernoulli {
02908
                   using type = typename FractionField<T>::template mul_t<</pre>
02909
                       typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02910
                       makefraction_t<T,</pre>
02911
                       typename T::template val<static_cast<typename T::inner_type>(-1)>,
02912
                       typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02913
02914
                   >;
02915
02916
                   template<typename floatType>
02917
                   static constexpr floatType value = type::template get<floatType>();
02918
              };
02919
02920
              template<typename T>
02921
               struct bernoulli<T, 0> {
02922
                  using type = typename FractionField<T>::one;
02923
02924
                   template<typename floatType>
02925
                   static constexpr floatType value = type::template get<floatType>();
02926
               };
02927
          } // namespace internal
02928
02932
          template<typename T, size_t n>
02933
          using bernoulli_t = typename internal::bernoulli<T, n>::type;
02934
          template<typename FloatType, typename T, size_t n >
inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02939
02940
02941
02942
           // bell numbers
02943
          namespace internal {
               template<typename T, size_t n, typename E = void>
02944
02945
               struct bell helper;
02946
02947
               template <typename T, size_t n>
02948
               struct bell_helper<T, n, std::enable_if_t<(n > 1)» {
02949
                   template<typename accum, size_t i, size_t stop>
02950
                   struct sum_helper {
02951
                    private:
02952
                       using left = typename T::template mul t<
```

```
combination_t<T, i, n-1>,
02954
                                   typename bell_helper<T, i>::type>;
02955
                      using new_accum = typename T::template add_t<accum, left>;
02956
                   public:
02957
                      using type = typename sum_helper<new_accum, i+1, stop>::type;
02958
                  };
02959
02960
                  template<typename accum, size_t stop>
02961
                  struct sum_helper<accum, stop, stop> {
02962
                      using type = accum;
02963
02964
02965
                  using type = typename sum helper<typename T::zero, 0, n>::type;
02966
              };
02967
02968
              {\tt template}{<}{\tt typename}\ {\tt T}{>}
              struct bell_helper<T, 0> {
02969
02970
                 using type = typename T::one;
02971
02972
02973
              template<typename T>
02974
              struct bell_helper<T, 1> {
02975
                  using type = typename T::one;
02976
02977
          } // namespace internal
02978
          template<typename T, size_t n>
02982
02983
          using bell_t = typename internal::bell_helper<T, n>::type;
02984
02988
          template<typename T, size_t n>
02989
          static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
02990
02991
          namespace internal {
02992
              template<typename T, int k, typename E = void>
02993
              struct alternate {};
02994
02995
              template<typename T, int k>
              struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
02996
02997
               using type = typename T::one;
                  static constexpr typename T::inner_type value = type::template get<typename</pre>
02998
     T::inner_type>();
02999
              };
03000
03001
              template<typename T, int k>
              struct alternate<T, k, std::enable_if_t<k % 2 != 0» {</pre>
03002
03003
                  using type = typename T::template sub_t<typename T::zero, typename T::one>;
03004
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
03005
              };
03006
          } // namespace internal
03007
03010
          template<typename T, int k>
03011
          using alternate_t = typename internal::alternate<T, k>::type;
03012
          template<typename T, size_t k>
03015
03016
          inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
03018
          namespace internal {
03019
              template<typename T, int n, int k, typename E = void>
03020
              struct stirling_1_helper {};
03021
03022
              template<typename T>
03023
              struct stirling_1_helper<T, 0, 0> {
03024
                 using type = typename T::one;
03025
              } ;
03026
03027
              template<typename T, int n>
              struct stirling_1_helper<T, n, 0, std::enable_if_t<(n > 0)» {
03028
03029
                  using type = typename T::zero;
03030
              };
03031
03032
              template<typename T, int n>
              struct stirling_1_helper<T, 0, n, std::enable_if_t<(n > 0)» {
03033
03034
                  using type = typename T::zero;
03035
03036
03037
              template<typename T, int n, int k>
03038
              struct stirling_1_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)» {
                  using type = typename T::template sub_t<
03039
                                   typename stirling_1_helper<T, n-1, k-1>::type,
typename T::template mul_t<</pre>
03040
03041
03042
                                       typename T::template inject_constant_t<n-1>,
03043
                                       typename stirling_1_helper<T, n-1, k>::type
03044
03045
          } // namespace internal
03046
03047
```

```
03052
           template<typename T, int n, int k>
03053
           using stirling_1_signed_t = typename internal::stirling_1_helper<T, n, k>::type;
03054
03059
           template<typename T, int n, int k>
           using stirling_1_unsigned_t = abs_t<typename internal::stirling_1_helper<T, n, k>::type>;
03060
03061
03066
           template<typename T, int n, int k>
03067
           static constexpr typename T::inner_type stirling_1_unsigned_v = stirling_1_unsigned_t<T, n, k>::v;
03068
          template<typename T, int n, int k>
static constexpr typename T::inner_type stirling_1_signed_v = stirling_1_signed_t<T, n, k>::v;
03073
03074
03075
03076
          namespace internal {
03077
               template<typename T, int n, int k, typename E = void>
03078
               struct stirling_2_helper {};
03079
03080
               template<typename T, int n>
               templatestypename 1, Int in>
struct stirling_2_helperTT, n, n, std::enable_if_t<(n >= 0)» {
    using type = typename T::one;
03081
03082
03083
               };
03084
03085
               template<typename T, int n>
               struct stirling_2_helper<T, n, 0, std::enable_if_t<(n > 0)» {
03086
03087
                   using type = typename T::zero;
03088
               };
03089
03090
               template<typename T, int n>
03091
               struct stirling_2_helper<T, 0, n, std::enable_if_t<(n > 0)» {
03092
                   using type = typename T::zero;
03093
03094
03095
               template<typename T, int n, int k>
03096
               struct stirling_2_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0) && (k < n)» {
03097
                   using type = typename T::template add_t<</pre>
                                     typename stirling_2_helper<T, n-1, k-1>::type,
typename T::template mul_t<</pre>
03098
03099
03100
                                          typename T::template inject constant t<k>,
03101
                                         typename stirling_2_helper<T, n-1, k>::type
03102
03103
           } // namespace internal
03104
03105
03110
           template<typename T, int n, int k>
03111
          using stirling_2_t = typename internal::stirling_2_helper<T, n, k>::type;
03112
03117
           template<typename T, int n, int k>
03118
           static constexpr typename T::inner_type stirling_2_v = stirling_2_t<T, n, k>::v;
03119
03120
          namespace internal {
03121
              template<typename T>
03122
               struct pow_scalar {
03123
                   template<size_t p>
03124
                    static constexpr DEVICE INLINED T func (const T& x) { return p == 0 ? static_cast<T>(1) :
03125
                       p % 2 == 0 ? func < p/2 > (x) * func < p/2 > (x) :
                        x * func<p/2>(x) * func<p/2>(x);
03126
03127
                   }
03128
               };
03129
03130
               template<typename T, typename p, size_t n, typename E = void>
03131
               requires IsEuclideanDomain<T>
03132
               struct pow;
03133
03134
               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<
03135
03136
03137
                        typename pow<T, p, n/2>::type,
03138
                        typename pow<T, p, n/2>::type
03139
                   >;
03140
               };
03141
03142
               template<typename T, typename p, size_t n>
0.3143
               struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)» {
0.3144
                   using type = typename T::template mul_t<</pre>
03145
                       p,
03146
                        typename T::template mul t<
03147
                            typename pow<T, p, n/2>::type,
03148
                            typename pow<T, p, n/2>::type
03149
03150
                   >;
03151
               }:
03152
03153
               template<typename T, typename p, size_t n>
03154
               struct pow<T, p, n, std::enable_if_t<n == 0» { using type = typename T::one; };</pre>
03155
          } // namespace internal
03156
          template<typename T, typename p, size_t n>
using pow_t = typename internal::pow<T, p, n>::type;
0.3161
03162
```

```
03168
          template<typename T, typename p, size_t n>
03169
          static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
03170
0.3171
          template<typename T, size_t p>
static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
03172
      internal::pow_scalar<T>::template func(x); }
03173
03174
          namespace internal {
03175
              template<typename, template<typename, size_t> typename, class>
03176
              struct make_taylor_impl;
03177
03178
              template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
              struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...» {
03179
03180
                 using type = typename polynomial<FractionField<T>::template val<typename coeff_at<T,
     Is>::type...>;
03181
             };
03182
          }
03183
03188
          template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
03189
          using taylor = typename internal::make_taylor_impl<</pre>
03190
              coeff_at,
0.3191
03192
              internal::make index sequence reverse<deg + 1>>::type;
03193
03194
          namespace internal {
              template<typename T, size_t i>
03195
03196
              struct exp_coeff {
03197
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03198
03199
03200
              template<typename T, size_t i, typename E = void>
03201
              struct sin_coeff_helper {};
03202
              template<typename T, size_t i>
03203
              struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
03204
03205
03206
03207
03208
              template<typename T, size_t i>
03209
              struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03210
                  using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03211
03212
03213
              template<typename T, size_t i>
03214
              struct sin_coeff {
03215
                 using type = typename sin_coeff_helper<T, i>::type;
03216
              };
03217
03218
              template<typename T, size_t i, typename E = void>
03219
              struct sh_coeff_helper {};
03220
03221
              template<typename T, size_t i>
03222
              struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0  {
                  using type = typename FractionField<T>::zero;
03223
03224
              };
03225
03226
              template<typename T, size_t i>
03227
              struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03228
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03229
03230
03231
              template<typename T, size_t i>
03232
              struct sh_coeff {
03233
                  using type = typename sh_coeff_helper<T, i>::type;
03234
03235
03236
              template<typename T, size_t i, typename E = void>
03237
              struct cos coeff helper {};
03238
03239
              template<typename T, size_t i>
03240
              struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1> {
                  using type = typename FractionField<T>::zero;
03241
03242
03243
03244
              template<typename T, size_t i>
03245
              struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03246
                 using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03247
              };
03248
03249
              template<typename T, size_t i>
03250
              struct cos_coeff {
03251
                  using type = typename cos_coeff_helper<T, i>::type;
03252
03253
03254
              template<typename T, size_t i, typename E = void>
03255
              struct cosh_coeff_helper {};
```

```
template<typename T, size_t i>
03257
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03258
                 using type = typename FractionField<T>::zero;
03259
03260
03261
03262
              template<typename T, size_t i>
03263
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0\times {
03264
                 using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03265
03266
              template<typename T, size_t i>
03267
03268
              struct cosh_coeff {
                  using type = typename cosh_coeff_helper<T, i>::type;
03269
03270
03271
03272
              template<typename T, size_t i>
03273
              struct geom_coeff { using type = typename FractionField<T>::one; };
03275
03276
              template<typename T, size_t i, typename E = void>
03277
              struct atan_coeff_helper;
03278
              template<typename T, size_t i>
struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03279
03280
                 using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>»;
03281
03282
03283
03284
              template<typename T, size_t i>
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03285
                  using type = typename FractionField<T>::zero;
03286
03287
03288
03289
               template<typename T, size_t i>
03290
              struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
03291
03292
              template<typename T, size_t i, typename E = void>
              struct asin_coeff_helper;
03293
03294
03295
              template<typename T, size_t i>
03296
               struct asin\_coeff\_helper<T, i, std::enable\_if\_t<(i \& 1) == 1> {
03297
                  using type = makefraction_t<T,
                       factorial_t<T, i - 1>,
03298
03299
                       typename T::template mul_t<
03300
                           typename T::template val<i>,
03301
                           T::template mul_t<
03302
                               pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
                               pow<T, factorial_t<T, i / 2>, 2
03303
03304
03305
03306
                       »;
03307
03308
              template<typename T, size_t i>
struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03309
03310
                  using type = typename FractionField<T>::zero;
03311
03312
03313
03314
              template<typename T, size_t i>
03315
              struct asin_coeff {
                  using type = typename asin_coeff_helper<T, i>::type;
03316
03317
03318
03319
              template<typename T, size_t i>
03320
              struct lnp1_coeff {
03321
                  using type = makefraction_t<T,
03322
                       alternate_t<T, i + 1>,
03323
                       typename T::template val<i>;
03324
03325
03326
              template<typename T>
03327
              struct lnpl_coeff<T, 0> { using type = typename FractionField<T>::zero; };
03328
              template<typename T, size_t i, typename E = void>
03329
03330
              struct asinh coeff helper;
03331
03332
               template<typename T, size_t i>
03333
               struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
                   using type = makefraction_t<T,</pre>
03334
                       typename T::template mul_t<
03335
                           alternate_t<T, i / 2>,
03336
                           factorial_t<T, i - 1>
03337
03338
03339
                       typename T::template mul_t<
03340
                           typename T::template mul_t<</pre>
03341
                               typename T::template val<i>,
03342
                               pow_t<T, factorial_t<T, i / 2>, 2>
```

```
pow_t<T, typename T::template inject_constant_t<4>, i / 2>
03344
03345
03346
                  >;
03347
              };
03348
03349
               template<typename T, size_t i>
03350
               struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03351
                 using type = typename FractionField<T>::zero;
03352
03353
              template<typename T, size_t i>
03354
03355
              struct asinh coeff {
03356
                  using type = typename asinh_coeff_helper<T, i>::type;
03357
03358
03359
              template<typename T, size_t i, typename E = void>
03360
              struct atanh_coeff_helper;
03361
03362
              template<typename T, size_t i>
03363
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03364
                   // 1/i
03365
                   using type = typename FractionField<T>:: template val<
03366
                       typename T::one,
03367
                       typename T::template inject_constant_t<i>;;
03368
              } ;
03369
03370
              template<typename T, size_t i>
03371
              struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03372
                  using type = typename FractionField<T>::zero;
03373
03374
03375
              template<typename T, size_t i>
03376
              struct atanh_coeff {
03377
                  using type = typename atanh_coeff_helper<T, i>::type;
03378
03379
03380
              template<typename T, size_t i, typename E = void>
03381
              struct tan_coeff_helper;
03382
              template<typename T, size_t i>
03383
              struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
    using type = typename FractionField<T>::zero;
03384
03385
03386
03387
03388
              template<typename T, size_t i>
03389
              struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
03390
              private:
                   // 4^((i+1)/2)
03391
                   using _4p = typename FractionField<T>::template inject_t<</pre>
03392
                       pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
03393
03394
                   // 4^((i+1)/2)
03395
                   using _4pm1 = typename FractionField<T>::template
     sub_t<_4p, typename FractionField<T>::one>;
    // (-1)^((i-1)/2)
03396
03397
                   using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
03398
                   using dividend = typename FractionField<T>::template mul_t<
03399
                       altp,
03400
                       FractionField<T>::template mul_t<</pre>
                       _4p,
03401
                       FractionField<T>::template mul_t<
03402
03403
                       _4pm1,
03404
                       bernoulli_t<T, (i + 1)>
03405
03406
03407
03408
               public:
                  using type = typename FractionField<T>::template div_t<dividend,</pre>
03409
03410
                       typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03411
              };
03412
03413
              template<typename T, size_t i>
03414
              struct tan_coeff {
03415
                  using type = typename tan_coeff_helper<T, i>::type;
03416
03417
03418
               template<typename T, size_t i, typename E = void>
03419
              struct tanh_coeff_helper;
03420
03421
              template<typename T, size t i>
              struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
03422
                  using type = typename FractionField<T>::zero;
03423
03424
03425
03426
              template<typename T, size_t i>
              struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
03427
03428
              private:
```

```
03429
                  using _4p = typename FractionField<T>::template inject_t<</pre>
                  pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
using _4pml = typename FractionField<T>::template
03430
03431
     sub_t<_4p, typename FractionField<T>::one>;
03432
                  using dividend =
03433
                      typename FractionField<T>::template mul_t<</pre>
03434
                           4p,
03435
                           typename FractionField<T>::template mul_t<</pre>
03436
                               _4pm1,
03437
                               bernoulli t<T, (i + 1) >>::type;
03438
              public:
               using type = typename FractionField<T>::template div_t<dividend,</pre>
03439
                      FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03440
03441
              };
03442
03443
              template<typename T, size_t i>
              using type = typename tanh_coeff_helper<T, i>::type;
};
03444
03445
03446
03447
          } // namespace internal
03448
03452
          template<typename Integers, size_t deg>
03453
          using exp = taylor<Integers, internal::exp_coeff, deg>;
03454
03458
          template<typename Integers, size_t deg>
          using expm1 = typename polynomial<FractionField<Integers>>::template sub_t<
03459
03460
              exp<Integers, deg>,
03461
              typename polynomial<FractionField<Integers>>::one>;
03462
03466
          template<typename Integers, size_t deg>
03467
          using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
03468
03472
          template<typename Integers, size_t deg>
03473
          using atan = taylor<Integers, internal::atan_coeff, deg>;
03474
03478
          template<typename Integers, size_t deg>
03479
          using sin = taylor<Integers, internal::sin_coeff, deg>;
03480
03484
          template<typename Integers, size_t deg>
03485
          using sinh = taylor<Integers, internal::sh_coeff, deg>;
03486
03491
          template<typename Integers, size_t deg>
03492
          using cosh = taylor<Integers, internal::cosh coeff, deg>;
03493
03498
          template<typename Integers, size_t deg>
03499
                     taylor<Integers, internal::cos_coeff, deg>;
03500
03505
          template<typename Integers, size_t deg>
          using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
03506
03507
03512
          template<typename Integers, size_t deg>
03513
          using asin = taylor<Integers, internal::asin_coeff, deg>;
03514
03519
          template<typename Integers, size_t deg>
03520
          using asinh = taylor<Integers, internal::asinh_coeff, deg>;
03521
03526
          template<typename Integers, size_t deg>
03527
          using atanh = taylor<Integers, internal::atanh_coeff, deg>;
03528
03533
          template<typename Integers, size_t deg>
          using tan = taylor<Integers, internal::tan_coeff, deg>;
03534
03535
03540
          template<typename Integers, size_t deg>
03541
          using tanh = taylor<Integers, internal::tanh_coeff, deg>;
03542 }
        // namespace aerobus
03543
03544 // continued fractions
03545 namespace aerobus {
03548
          template<int64_t... values>
03549
          struct ContinuedFraction {};
03550
03553
          template<int64_t a0>
          struct ContinuedFraction<a0> {
03554
03556
              using type = typename q64::template inject_constant_t<a0>;
03558
              static constexpr double val = static_cast<double>(a0);
03559
03560
          template<int64_t a0, int64_t... rest>
03564
03565
          struct ContinuedFraction<a0, rest...> {
              using type = q64::template add_t<
03567
03568
                      typename q64::template inject_constant_t<a0>,
03569
                      typename q64::template div_t<
03570
                           typename q64::one,
03571
                           typename ContinuedFraction<rest...>::type
03572
03573
03575
              static constexpr double val = type::template get<double>():
```

```
03576
                 };
03577
03581
                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>;
03583
               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 =
         03587
               using SQRT3_fraction =
         ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 
          // NOLINT
03588 } // namespace aerobus
03589
03590 // known polynomials
03591 namespace aerobus {
03592
                 // CChebyshev
03593
                namespace internal {
03594
                       template<int kind, size_t deg, typename I>
03595
                       struct chebyshev_helper {
03596
                              using type = typename polynomial<I>::template sub_t<
03597
                                     typename polynomial<I>::template mul_t<</pre>
03598
                                            typename polynomial<I>::template mul_t<</pre>
03599
                                                  typename polynomial<I>::template inject_constant_t<2>,
                                                  typename polynomial<I>::X>,
03600
03601
                                            typename chebyshev_helper<kind, deg - 1, I>::type
03602
03603
                                     typename chebyshev_helper<kind, deg - 2, I>::type
03604
03605
                       };
03606
03607
                       template<tvpename I>
03608
                       struct chebyshev_helper<1, 0, I> {
03609
                             using type = typename polynomial<I>::one;
03610
03611
03612
                       template<typename I>
                       struct chebyshev_helper<1, 1, I> {
03613
03614
                              using type = typename polynomial<I>::X;
03615
03616
03617
                       template<typename I>
                       struct chebyshev_helper<2, 0, I> {
03618
03619
                             using type = typename polynomial<I>::one;
03620
03621
03622
                        template<typename I>
03623
                        struct chebyshev_helper<2, 1, I> {
                              using type = typename polynomial<I>::template mul_t<</pre>
03624
                                     typename polynomial<I>::template inject_constant_t<2>,
03625
03626
                                     typename polynomial<I>::X>;
03627
                        };
03628
                 } // namespace internal
03629
03630
                 // Laguerre
03631
                namespace internal {
03632
                       template<size t deg, typename I>
03633
                       struct laguerre_helper {
03634
                              using Q = FractionField<I>;
03635
                              using PQ = polynomial<Q>;
03636
03637
                         private:
                              // Lk = (1 / k) * ((2 * k - 1 - x) * 1km1 - (k - 2)Lkm2)
03638
03639
                              using lnm2 = typename laguerre_helper<deg - 2, I>::type;
                               using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03640
03641
                               // -x + 2k-1
03642
                               using p = typename PQ::template val<
03643
                                    typename Q::template inject_constant_t<-1>,
                                     typename Q::template inject_constant_t<2 * deg - 1»;
03644
03645
                               // 1/n
03646
                              using factor = typename PQ::template inject_ring_t<</pre>
                                     typename Q::template val<typename I::one, typename I::template
03647
         inject_constant_t<deg>>;
03648
03649
                         public:
03650
                              using type = typename PO::template mul t <
03651
                                     factor,
03652
                                      typename PQ::template sub_t<
03653
                                            typename PQ::template mul_t<</pre>
03654
03655
                                                   1 nm1
03656
03657
                                            typename PQ::template mul_t<
03658
                                                   typename PQ::template inject_constant_t<deg-1>,
03659
                                                   1 nm2
03660
03661
03662
                              >;
```

```
} ;
03664
03665
              template<typename I>
              struct laguerre_helper<0, I> {
03666
                 using type = typename polynomial<FractionField<I»::one;
03667
03668
              };
03669
03670
              template<typename I>
03671
              struct laguerre_helper<1, I> {
              private:
03672
                  using PQ = polynomial<FractionField<I»;
03673
03674
               public:
03675
                 using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
03676
03677
          } // namespace internal
03678
          // Bernstein
03679
03680
         namespace internal {
03681
             template<size_t i, size_t m, typename I, typename E = void>
03682
              struct bernstein_helper {};
03683
03684
              template<typename I>
              struct bernstein_helper<0, 0, I> {
03685
                  using type = typename polynomial<I>::one;
03686
03687
03688
03689
              template<size_t i, size_t m, typename I>
              03690
03691
03692
               private:
                 using P = polynomial<I>;
03693
03694
               public:
03695
                 using type = typename P::template mul_t<</pre>
03696
                          typename P::template sub_t<typename P::one, typename P::X>,
03697
                          typename bernstein_helper<i, m-1, I>::type>;
03698
              };
03699
03700
              template<size_t i, size_t m, typename I>
              struct bernstein_helper<i, m, I, std::enable_if_t<
(m > 0) && (i == m)» {
03701
03702
               private:
03703
                  using P = polynomial<I>;
03704
03705
               public:
03706
                  using type = typename P::template mul_t<
03707
                          typename P::X,
03708
                          typename bernstein_helper<i-1, m-1, I>::type>;
03709
03710
              template<size_t i, size_t m, typename I>
03711
              struct bernstein_helper<i, m, I, std::enable_if_t<
(m > 0) && (i > 0) && (i < m) » {
03712
03713
03714
               private:
03715
                 using P = polynomial<I>;
               nublic:
03716
03717
                  using type = typename P::template add_t<
03718
                          typename P::template mul_t<
03719
                              typename P::template sub_t<typename P::one, typename P::X>,
03720
                               typename bernstein_helper<i, m-1, I>::type>,
03721
                          typename P::template mul_t<
03722
                              typename P::X,
                              typename bernstein_helper<i-1, m-1, I>::type»;
03723
03724
          };
} // namespace internal
03725
03726
03727
          // AllOne polynomials
          namespace internal {
03728
03729
              template<size_t deg, typename I>
              struct AllOneHelper {
03730
03731
                  using type = aerobus::add_t<
03732
                      typename polynomial<I>::one,
03733
                      typename aerobus::mul_t<</pre>
03734
                          typename polynomial<I>::X,
03735
                          typename AllOneHelper<deg-1, I>::type
03736
03737
              };
03738
03739
              template<typename I>
03740
              struct AllOneHelper<0, I> {
03741
                  using type = typename polynomial<I>::one;
03742
03743
          } // namespace internal
03744
03745
          // Bessel polynomials
          namespace internal {
03746
03747
             template<size_t deg, typename I>
03748
              struct BesselHelper {
03749
              private:
```

```
using P = polynomial<I>;
03751
                  using factor = typename P::template monomial_t<</pre>
03752
                       typename I::template inject_constant_t<(2*deg - 1)>,
                      1>;
03753
03754
               public:
03755
                  using type = typename P::template add_t<
03756
                       typename P::template mul_t<
03757
                           factor,
03758
                           typename BesselHelper<deg-1, I>::type
03759
03760
                       typename BesselHelper<deg-2, I>::type
03761
                  >;
03762
              };
03763
03764
               template<typename I>
03765
              struct BesselHelper<0, I> {
03766
                  using type = typename polynomial<I>::one;
03767
03768
03769
              template<typename I>
03770
              struct BesselHelper<1, I> {
03771
               private:
03772
                  using P = polynomial<I>;
03773
               public:
03774
                  using type = typename P::template add_t<
03775
                    typename P::one,
                      typename P::X
03776
03777
03778
              } ;
03779
          } // namespace internal
03780
03781
          namespace known polynomials {
03783
              enum hermite_kind {
03785
                  probabilist,
03787
                  physicist
              };
03788
03789
          }
03790
03791
03792
          namespace internal {
03793
              template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03794
              struct hermite_helper {};
03795
03796
              template<size_t deg, typename I>
03797
              struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
03798
               private:
03799
                  using hnm1 = typename hermite_helper<deg - 1,
      known_polynomials::hermite_kind::probabilist, I>::type;
03800
                  using hnm2 = typename hermite_helper<deg - 2,
      known polynomials::hermite kind::probabilist, I>::type;
03801
03802
               public:
03803
                  using type = typename polynomial<I>::template sub_t<</pre>
                       typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnml>,
typename polynomial<I>::template mul_t<</pre>
03804
03805
                           typename polynomial<I>::template inject_constant_t<deg - 1>,
03806
                           hnm2
03808
03809
03810
              };
03811
03812
              template<size_t deg, typename I>
03813
              struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03814
03815
                  using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
      I>::type;
03816
                  using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
      I>::tvpe;
03817
03818
               public:
03819
                  using type = typename polynomial<I>::template sub_t<</pre>
03820
                       // 2X Hn-1
03821
                       typename polynomial<I>::template mul_t<</pre>
03822
                           typename pi64::val<typename I::template inject_constant_t<2>,
                           typename I::zero>, hnm1>,
03823
03824
03825
                       typename polynomial<I>::template mul_t<
03826
                           typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03827
                           hnm2
03828
03829
                  >;
03830
              } ;
03831
03832
               template<typename I>
03833
               struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03834
                  using type = typename polynomial<I>::one;
03835
              };
```

```
03837
              template<typename I>
03838
              struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03839
                 using type = typename polynomial<I>::X;
03840
03841
03842
              template<typename I>
03843
              struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03844
                using type = typename pi64::one;
03845
03846
03847
              template<tvpename I>
03848
              struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03849
03850
                  using type = typename polynomial<I>::template val<</pre>
03851
                      typename I::template inject_constant_t<2>,
                      typename I::zero>;
03852
03853
03854
          } // namespace internal
03855
03856
          // legendre
03857
          namespace internal {
03858
              template<size_t n, typename I>
03859
              struct legendre_helper {
03860
               private:
                  using Q = FractionField<I>;
03861
03862
                  using PQ = polynomial<Q>;
                  // 1/n constant
// (2n-1)/n X
03863
03864
                  using fact_left = typename PQ::template monomial_t<</pre>
03865
03866
                      makefraction_t<I,
03867
                          typename I::template inject_constant_t<2*n-1>,
03868
                           typename I::template inject_constant_t<n>
03869
                      >,
                  1>;
// (n-1) / n
03870
03871
03872
                  using fact_right = typename PQ::template val<
                      makefraction_t<I,
03874
                          typename I::template inject_constant_t<n-1>,
03875
                           typename I::template inject_constant_t<n>>;
03876
03877
               public:
                  using type = PQ::template sub_t<
03878
03879
                           typename PQ::template mul_t<
03880
                               fact_left,
03881
                               typename legendre_helper<n-1, I>::type
03882
                           typename PQ::template mul_t<</pre>
03883
03884
                               fact_right,
03885
                               typename legendre helper<n-2, I>::type
03886
03887
                      >;
03888
03889
03890
              template<typename I>
03891
              struct legendre_helper<0, I> {
03892
                 using type = typename polynomial<FractionField<I>::one;
03893
03894
03895
              template<typename I>
03896
              struct legendre_helper<1, I> {
                  using type = typename polynomial<FractionField<I>::X;
03897
03898
03899
          } // namespace internal
03900
03901
          // bernoulli polynomials
03902
          namespace internal {
03903
              template<size t n>
              struct bernoulli_coeff {
03904
                  template<typename T, size_t i>
03905
03906
                  struct inner {
03907
                   private:
03908
                      using F = FractionField<T>;
03909
                   public:
03910
                      using type = typename F::template mul_t<</pre>
03911
                           typename F::template inject_ring_t<combination_t<T, i, n»,
03912
                           bernoulli_t<T, n-i>
03913
03914
                  };
03915
              }:
          } // namespace internal
03916
03917
03918
          namespace internal {
03919
              template<size_t n>
03920
              struct touchard_coeff {
03921
                  template<typename T, size_t i>
03922
                  struct inner {
```

```
using type = stirling_2_t<T, n, i>;
03924
                  };
03925
          } // namespace internal
03926
03927
03928
          namespace internal {
03929
              template<typename I = aerobus::i64>
03930
              struct AbelHelper {
               private:
03931
                  using P = aerobus::polynomial<I>;
03932
03933
03934
                public:
03935
                  // to keep recursion working, we need to operate on a*n and not just a
03936
                   template<size_t deg, I::inner_type an>
03937
                   struct Inner {
03938
                       // abel(n, a) = (x-an) * abel(n-1, a)
03939
                       using type = typename aerobus::mul_t<
                           typename Inner<deg-1, an>::type,
typename aerobus::sub_t<typename P::X, typename P::template inject_constant_t<an>>
03940
03941
03942
                       >;
03943
                   };
03944
                  // abel(0, a) = 1
template<I::inner_type an>
03945
03946
                  struct Inner<0, an> {
    using type = P::one;
03947
03948
03949
03950
03951
                  // abel(1, a) = X
03952
                   template<I::inner_type an>
                   struct Inner<1, an>
03953
03954
                       using type = P::X;
03955
03956
03957
          } // namespace internal
03958
03960
          namespace known polynomials {
03961
03970
               template<size_t n, auto a, typename I = aerobus::i64>
03971
              using abel = typename internal::AbelHelper<I>::template Inner<n, a*n>::type;
03972
03980
               template <size_t deg, typename I = aerobus::i64>
03981
              using chebyshev T = typename internal::chebyshev helper<1, deg, I>::type;
03982
03992
               template <size_t deg, typename I = aerobus::i64>
03993
               using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
03994
               template <size_t deg, typename I = aerobus::i64>
04004
04005
              using laguerre = typename internal::laguerre_helper<deg, I>::type;
04006
04013
              template <size_t deg, typename I = aerobus::i64>
              using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
04014
     I>::type;
04015
              template <size_t deg, typename I = aerobus::i64>
04022
04023
              using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
04024
04035
               template<size_t i, size_t m, typename I = aerobus::i64>
04036
              using bernstein = typename internal::bernstein_helper<i, m, I>::type;
04037
04047
              template<size_t deg, typename I = aerobus::i64>
              using legendre = typename internal::legendre_helper<deg, I>::type;
04048
04049
04059
               template<size_t deg, typename I = aerobus::i64>
04060
              using bernoulli = taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
04061
04068
               template<size_t deg, typename I = aerobus::i64>
              using allone = typename internal::AllOneHelper<deg, I>::type;
04069
04070
04078
               template<size_t deg, typename I = aerobus::i64>
04079
               using bessel = typename internal::BesselHelper<deg, I>::type;
04080
              template<size_t deg, typename I = aerobus::i64>
using touchard = taylor<I, internal::touchard_coeff<deg>::template inner, deg>;
04088
04089
04090
             // namespace known_polynomials
04091 } // namespace aerobus
04092
04093
04094 #ifdef AEROBUS CONWAY IMPORTS
04095
04096 // conway polynomials
04097 namespace aerobus {
04101
          template<int p, int n>
04102
          struct ConwayPolynomial {};
04103
04104 #ifndef DO NOT DOCUMENT
          #define ZPZV ZPZ::template val
04105
```

```
#define POLYV aerobus::polynomial<ZPZ>::template val
                      template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<1>; }; // NOLINT
 04108
                                       template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
                        POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
04109
                                        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 =
 04110
                        template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
 04111
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>; ZPZV<1>; ZPZV<2>; (NOLINT template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
 04112
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2, ZPZV<1>, ZPZV<2, ZPZV<1>, ZPZV<2, ZPZV<1>, ZPZV<1
, 
 04113
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
 04114
                                       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<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>; }; //
 04115
                        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 , Z
                        ZPZV<1»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<2, 11> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
                        ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<2, 12> { using ZPZ = aerobus::zpz<2>; using type =
04118
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                       template<> struct ConwayPolynomial<2, 13> { using ZPZ = aerobus::zpz<2>; using type =
04119
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>,
                        ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type =
04120
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>,
                        ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>; }; // NOLINT
template<> struct ConwayPolynomial<2, 15> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
04121
                                         template<> struct ConwayPolynomial<2, 16> { using ZPZ = aerobus::zpz<2>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0, ZPZV<1»; }; // NOLINT
    template<> struct ConwayPolynomial<2, 17> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04123
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<2, 18> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                        template<> struct ConwayPolynomial<2, 19> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
                        template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                         ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1</pre>; };
                        // NOLINT
                                         template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
04127
                        POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
                                      template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
                        POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2»; }; // NOLINT
 04129
                                        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 =
04130
                        POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; };
                                                                                                                                                                                                                                                 // NOLINT
                                         template<> struct ConwayPolynomial<3, 5> { using ZPZ = aerobus::zpz<3>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT
 04132
                                      template<> struct ConwayPolynomial<3, 6> { using ZPZ = aerobus::zpz<3>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<3, 7> { using ZPZ = aerobus::zpz<3>; using type =
04133
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<1»; }; // NOLINT
 04134
                                         template<> struct ConwayPolynomial<3, 8> { using ZPZ = aerobus::zpz<3>; using type :
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2»; }; // NOLINT
 04135
                                     template<> struct ConwayPolynomial<3, 9> { using ZPZ = aerobus::zpz<3>; using type :
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1>; }; //
                        NOLINT
                                       template<> struct ConwayPolynomial<3, 10> { using ZPZ = aerobus::zpz<3>; using type
04136
                        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>
04137
                                       template<> struct ConwayPolynomial<3, 11> { using ZPZ = aerobus::zpz<3>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>; }; // NOLINT template<> struct ConwayPolynomial<3, 12> { using ZPZ = aerobus::zpz<3>; using type =
04138
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                      template<> struct ConwayPolynomial<3, 13> { using ZPZ = aerobus::zpz<3>; using type =
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT template<> struct ConwayPolynomial<3, 14> { using ZPZ = aerobus::zpz<3>; using type =
 04140
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<1>,
                               ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<3, 15> { using ZPZ = aerobus::zpz<3>; using type =
                               Template(> Struct ConwayFolynomial(3, 15) { using ZFZ - derobus::2p2<3>, using type - POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
04142
                                 ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<3, 17> { using ZPZ = aerobus::zpz<3>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZ
04144
                                 ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0</pre>
                                                  template<> struct ConwayPolynomial<3, 19> { using ZPZ = aerobus::zpz<3>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1»; }; //
                                NOLINT
                                                     template<> struct ConwayPolynomial<3, 20> { using ZPZ = aerobus::zpz<3>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>
                                 ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2</pre>
                                 // NOLINT
04147
                                                      template<> struct ConwayPolynomial<5, 1> { using ZPZ = aerobus::zpz<5>; using type =
                               POLYV<ZPZV<1>, ZPZV<3»; }; // NOLINT
04148
                                                       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 =
04149
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
04150
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; };
                                                                                                                                                                                                                                                                                                                                       // NOLINT
                                                    template<> struct ConwayPolynomial<5, 5> { using ZPZ = aerobus::zpz<5>; using type =
04151
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<3»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<5, 6> { using ZPZ = aerobus::zpz<5>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<5, 7> { using ZPZ = aerobus::zpz<5>; using type =
04153
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
04154
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<5, 9> { using ZPZ = aerobus::zpz<5>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<3»; }; //
                              \label{eq:convayPolynomial} $$ template<> struct ConwayPolynomial<5, 10> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<1>, ZPZV<1>, ZPZV<5, ZPZV<5>, ZPZV<5, ZPZV<
04156
                                ZPZV<2»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<5, 11> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
                                ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                                                      template<> struct ConwayPolynomial<5, 12> { using ZPZ = aerobus::zpz<5>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<2»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<5, 13> { using ZPZ = aerobus::zpz<5>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<3»; }; // NOLINT</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<0>, ZPZV<4>, ZPZV<4>,
04160
                               ZPZV<2>, ZPZV<3>, ZPZV<0>, ZPZV<1>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<5, 15> { using ZPZ = aerobus::zpz<5>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<3»; }; // NOLINT
04162
                                                       template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4 , ZPZV<4
                                                      template<> struct ConwayPolynomial<5, 17> { using ZPZ = aerobus::zpz<5>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<3»; }; // NOLINT</pre>
04164
                                                  template<> struct ConwayPolynomial<5, 18> { using ZPZ = aerobus::zpz<5>; using type =
                               POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<1>, 2PZV<1
                                ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<2», }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<5, 19> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
04165
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, 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<
                                ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<0>, ZPZV<4>, ZPZV<1>, ZPZV<2>; };
04167
                                                     template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
                               POLYV<ZPZV<1>, ZPZV<4»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
                              POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<7, 3> { using ZPZ = aerobus::zpz<7>; using type =
04169
                             POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4»; }; // NOLINT template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; }; // NOLINT
04171
                                                   template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT template<> struct ConwayPolynomial<7, 6> { using ZPZ = aerobus::zpz<7>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<4>, ZPZV<6>, ZPZV<3»; };
                                       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 , ZP
                                                                  template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
  04174
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<6>, ZPZV<2>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<7, 9> { using ZPZ = aerobus::zpz<7>; using type =
 04175
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
 04176
                                                                  template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<3>,
                                          ZPZV<3»; }; // NOLINT
                                                                  template<> struct ConwayPolynomial<7, 11> { using ZPZ = aerobus::zpz<7>; using type =
 04177
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT
                                                                 template<> struct ConwayPolynomial<7, 12> { using ZPZ = aerobus::zpz<7>; using type
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<5>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
 04179
                                                                   template<> struct ConwayPolynomial<7, 13> { using ZPZ = aerobus::zpz<7>; using type
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<0>, ZPZV<6>, ZPZV<0>, ZPZV<4»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<7, 14> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6 , Z
                                         ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<3»; }; // NOLINT
  template<> struct ConwayPolynomial<7, 15> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
 04181
                                          ZPZV<6>, ZPZV<6>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<4»; }; // NOLINT</pre>
                                                                  template<> struct ConwayPolynomial<7, 16> { using ZPZ = aerobus::zpz<7>; using type =
 04182
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<5>,
                                          04183
                                         template<> struct ConwayPolynomial<7, 17> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
                                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT
                                                                     template<> struct ConwayPolynomial<7, 18> { using ZPZ = aerobus::zpz<7>; using type =
 04184
                                          POLYV<ZPZV<1>, ZPZV<0>, 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»; }; // NOLINT</pre>
 04185
                                         template<> struct ConwayPolynomial<7, 19> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
                                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<4»; }; //</pre>
                                         template<> struct ConwayPolynomial<7, 20> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                            ZPZV<2>, ZPZV<5>, ZPZV<2>, ZPZV<3>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<3>, ZPZV<3>
                                            // NOLINT
                                                                     template<> struct ConwayPolynomial<11, 1> { using ZPZ = aerobus::zpz<11>; using type =
                                         POLYV<ZPZV<1>, ZPZV<9»; }; // NOLINT
  04188
                                                                     template<> struct ConwayPolynomial<11, 2> { using ZPZ = aerobus::zpz<11>; using type =
                                         POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2»; }; // NOLINT
 04189
                                                                   template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type =
  04190
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
                                                                     template<> struct ConwayPolynomial<11, 5> { using ZPZ = aerobus::zpz<11>; using type =
  04191
                                         template<> struct ConwayPolynomial<11, 6> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<6>, ZPZV<7>, ZPZV<2»; }; // NOLINT</pre>
  04192
                                                                     template<> struct ConwayPolynomial<11, 7> { using ZPZ = aerobus::zpz<11>; using type
  04193
                                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<9»; }; // NOLINT
                                                                 template<> struct ConwayPolynomial<11, 8> { using ZPZ = aerobus::zpz<11>; using type =
  04194
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, 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<0>, ZPZV<9>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<8</pre>
  04195
                                         NOLINT
 04196
                                                                    template<> struct ConwayPolynomial<11, 10> { using ZPZ = aerobus::zpz<11>; using type =
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<10>, ZPZV<6>, ZPZV<6>,
                                          ZPZV<2»; }; // NOLINT</pre>
 04197
                                                                 template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<10>, ZPZV<9»; }; // NOLINT
                                                                   template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type =
                                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6 , ZPZV<6
                                                                 template<> struct ConwayPolynomial<11, 13> { using ZPZ = aerobus::zpz<11>; using type =
 04199
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         template<> struct ConwayPolynomial<11, 14> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<5>, ZPZV<5 , ZPZV<5 ,
04200
                                          ZPZV<4>, ZPZV<8>, ZPZV<6>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
 04201
                                                                   template<> struct ConwayPolynomial<11, 15> { using ZPZ = aerobus::zpz<11>; using type =
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<7>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<9»; }; // NOLINT
                                          template<> struct ConwayPolynomial<11, 16> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
 04202
                                          ZPZV<1>, ZPZV<3>, ZPZV<5>, ZPZV<3>, ZPZV<10>, ZPZV<9>, ZPZV<2»; }; // NOLINT</pre>
  04203
                                                                  template<> struct ConwayPolynomial<11, 17> { using ZPZ = aerobus::zpz<11>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
  04204
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<8>, ZPZV<10>, ZPZV<8>,
                            \texttt{ZPZV<3>, ZPZV<9>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<9>, ZPZV<8>, ZPZV<2>, ZPZV<2>, ZPZV<2»; }; // \texttt{NOLINT} 
                                          template<> struct ConwayPolynomial<11, 19> { using ZPZ = aerobus::zpz<11>; using type :
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<2>, ZPZV<9»; }; //</pre>
                           NOLINT
                          template<> struct ConwayPolynomial<11, 20> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
                            ZPZV<9>, ZPZV<1>, ZPZV<5>, ZPZV<7>, ZPZV<2>, ZPZV<4>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<5</pre>; };
                           // NOLINT
04207
                                            template<> struct ConwayPolynomial<13, 1> { using ZPZ = aerobus::zpz<13>; using type =
                          POLYV<ZPZV<1>, ZPZV<11»; }; // NOLINT
                                            template<> struct ConwayPolynomial<13, 2> { using ZPZ = aerobus::zpz<13>; using type =
04208
                          POLYV<ZPZV<1>, ZPZV<12>, ZPZV<2»; }; // NOLINT
 04209
                                          template<> struct ConwayPolynomial<13, 3> { using ZPZ = aerobus::zpz<13>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<13, 4> { using ZPZ = aerobus::zpz<13>; using type =
04210
                          POLYVCZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2; }; // NOLINT
template<> struct ConwayPolynomial<13, 5> { using ZPZ = aerobus::zpz<13>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<11»; }; // NOLINT
                                             template<> struct ConwayPolynomial<13, 6> { using ZPZ = aerobus::zpz<13>; using type =
 04212
                          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 =
04213
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<1>; ); // NOLINT template<> struct ConwayPolynomial<13, 8> { using ZPZ = aerobus::zpz<13>; using type =
04214
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<12>, ZPZV<2>, ZPZV<3>, ZPZV<2»; };
 04215
                                          template<> struct ConwayPolynomial<13, 9> { using ZPZ = aerobus::zpz<13>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<8>, ZPZV<12>, ZPZV<12>, ZPZV<11>; };
                            // NOLINT
04216
                                           template<> struct ConwayPolynomial<13, 10> { using ZPZ = aerobus::zpz<13>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>,
                           ZPZV<2»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<13, 11> { using ZPZ = aerobus::zpz<13>; using type
 04217
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<3>, ZPZV<11»; }; // NOLINT</pre>
04218
                                             template<> struct ConwayPolynomial<13, 12> { using ZPZ = aerobus::zpz<13>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<8>, ZPZV<8>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4 , ZPZV<4
                                             template<> struct ConwayPolynomial<13, 13> { using ZPZ = aerobus::zpz<13>; using type
                         Cemplate<> Struct ComwayFolynomialx13, 13> { using 2PZ - aerObus::2pZ<13>; using type -
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
04220
                           ZPZV<11>, ZPZV<7>, ZPZV<10>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>,
                          ZPZV<2>, ZPZV<11>, ZPZV<10>, ZPZV<11>, ZPZV<8>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<13, 16> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZV<0>; ZPZV<0>; ZPZV<0>, ZPZV<0>; ZPZV<0 ; 
                                             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<6+, ZPZV<11»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                        // NOLINT
                          template<> struct ConwayPolynomial<13, 18> { using ZPZ = aerobus::zpz<13>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<11>,
04224
                           ZPZV<11>, ZPZV<9>, ZPZV<5>, ZPZV<3>, ZPZV<5>, ZPZV<6>, ZPZV<0>, ZPZV<9>, ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<13, 19> { using ZPZ = aerobus::zpz<13>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<9>, ZPZV<11»; };</pre>
                           NOLINT
04226
                          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<1>, ZPZV<1 , ZPZ
                            ZPZV<9>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<7>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5</pre>; };
                                          template<> struct ConwayPolynomial<17, 1> { using ZPZ = aerobus::zpz<17>; using type =
04227
                          POLYV<ZPZV<1>, ZPZV<14»; }; // NOLINT
                                            template<> struct ConwayPolynomial<17, 2> { using ZPZ = aerobus::zpz<17>; using type =
 04228
                          POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT
                                             template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
 04230
                          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
 04231
                                            template<> struct ConwayPolynomial<17, 6> { using ZPZ = aerobus::zpz<17>; using type =
04232
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<10>, ZPZV<3>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<17, 7> { using ZPZ = aerobus::zpz<17>; using type
 04233
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<14»; }; // NOLINT
04234
                                            template<> struct ConwayPolynomial<17, 8> { using ZPZ = aerobus::zpz<17>; using type =
                         POLYV-ZPZV-1>, ZPZV-(>), ZPZV-(>), ZPZV-(>), ZPZV-(>), ZPZV-1>, ZPZV-12>, ZPZV-6>, Z
 04235
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<8>, ZPZV<14»; };
 04236
                                          template<> struct ConwayPolynomial<17, 10> { using ZPZ = aerobus::zpz<17>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<6>, ZPZV<5>, ZPZV<9>, ZPZV<12>, ZPZV<3»; }; // NOLINT
 04237
                                          template<> struct ConwavPolvnomial<17, 11> { using ZPZ = aerobus::zpz<17>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<5>, ZPZV<14»; }; // NOLINT
   template<> struct ConwayPolynomial<17, 12> { using ZPZ = aerobus::zpz<17>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<15, ZPZV<6>,
                               ZPZV<14>, ZPZV<9>, ZPZV<3»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                  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<1>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>; // NOLINT template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::zpz<17>; using type =
04241
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<16>, ZPZV<6>, ZPZV<14>, ZPZV<14»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<17, 16> { using ZPZ = aerobus::zpz<17>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<13>, ZPZV<13>, ZPZV<13>, ZPZV<13>, ZPZV<12>, ZPZV<13>, ZPZ
04243
                               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>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<5, ZPZV<1>, ZPZV<5, ZPZV<5
                              ZPZV<7>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<11>, ZPZV<13>, ZPZV<13>, ZPZV<9>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<17, 19> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04245
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<14»; }; //</pre>
04246
                                                  template<> struct ConwayPolynomial<17, 20> { using ZPZ = aerobus::zpz<17>; using type
                              template<> struct Conwayrolynomial<11, 20> { using 2r2 - derobus::2p2\17, using cype - POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
                               ZPZV<3»; }; // NOLINT
04247
                                                     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 =
04248
                              POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2»; }; // NOLINT
04249
                                                    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 =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<19, 5> { using ZPZ = aerobus::zpz<19>; using type =
04251
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<19, 6> { using ZPZ = aerobus::zpz<19>; using type =
04252
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<2, ZPZ
04253
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<17»; }; //
04254
                                                    template<> struct ConwayPolynomial<19, 8> { using ZPZ = aerobus::zpz<19>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<10>, ZPZV<3>, ZPZV<3>, ZPZV<2»; };
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  // NOLINT
04255
                                                  template<> struct ConwayPolynomial<19, 9> { using ZPZ = aerobus::zpz<19>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<16>, ZPZV<17»; };
                                // NOLINT
04256
                                                    template<> struct ConwayPolynomial<19, 10> { using ZPZ = aerobus::zpz<19>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<13>, ZPZV<17>, ZPZV<3>, ZPZV<4>,
                               ZPZV<2»; }; // NOLINT</pre>
04257
                                                    template<> struct ConwayPolynomial<19, 11> { using ZPZ = aerobus::zpz<19>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                     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</pre>
04259
                                                     template<> struct ConwayPolynomial<19, 13> { using ZPZ = aerobus::zpz<19>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                    template<> struct ConwayPolynomial<19, 14> { using ZPZ = aerobus::zpz<19>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<11>, ZPZV<11>,
                                ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<7>, ZPZV<2»; }; // NOLINT</pre>
04261
                                                 template<> struct ConwayPolynomial<19, 15> { 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<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>; // NOLINT template<> struct ConwayPolynomial<19, 16> { using ZPZ = aerobus::zpz<19>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<13>, ZPZV<0>, ZPZV<15>, ZPZV<9>, ZPZV<6>, ZPZV<14>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<19, 17> { using ZPZ = aerobus::zpz<19>; using type =
04263
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3, ZPZV
                                                    template<> struct ConwayPolynomial<19, 18> { using ZPZ = aerobus::zpz<19>; using type
04264
                               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<5>, ZPZV<5 , ZPZ
                               ZPZV<17>, ZPZV<5>, ZPZV<0>, ZPZV<16>, ZPZV<5>, ZPZV<7>, ZPZV<3>, ZPZV<14>, ZPZV<2»; };
                              template<> struct ConwayPolynomial<19, 19> { using ZPZ = aerobus::zpz<19>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<17»; };</pre>
                               NOLINT
                                                    template<> struct ConwayPolynomial<19, 20> { using ZPZ = aerobus::zpz<19>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<13>, ZPZV<0>, 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
```

```
04268
                                     template<> struct ConwayPolynomial<23, 2> { using ZPZ = aerobus::zpz<23>; using type =
                     POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; }; // NOLINT
                                 template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; }; // NOLINT
                   template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<18»; }; // NOLINT
04272
                                 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 =
04273
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<18»; }; // NOLINT
04274
                                   template<> struct ConwayPolynomial<23, 8> { using ZPZ = aerobus::zpz<23>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, // NOLINT
                                 template<> struct ConwayPolynomial<23, 9> { using ZPZ = aerobus::zpz<23>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<8>, ZPZV<9>, ZPZV<18»; };
                     // NOLINT
                     template<> struct ConwayPolynomial<23, 10> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<15>, ZPZV<5>, ZPZV<6>, ZPZV<1>,
04276
                     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>,
                     ZPZV<7>, ZPZV<18»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<23, 12> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<21>, ZPZV<15>, ZPZV<14>, ZPZV<14>, ZPZV<12>, ZPZV<15>; }; // NOLINT
                                 template<> struct ConwayPolynomial<23, 13> { using ZPZ = aerobus::zpz<23>; using type =
04279
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<18»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<23, 14> { using ZPZ = aerobus::zpz<23>; using type =
04280
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<1>, ZPZV<5>, ZPZV<1>,
                     ZPZV<18>, ZPZV<19>, ZPZV<1>, ZPZV<22>, ZPZV<5»; };</pre>
                                                                                                                                                                                                             // NOLINT
                                   template<> struct ConwayPolynomial<23, 15> { using ZPZ = aerobus::zpz<23>; using type =
04281
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<8>, ZPZV<15>, ZPZV<9>, ZPZV<7>, ZPZV<18>, ZPZV<18*; }; // NOLINT
    template<> struct ConwayPolynomial<23, 16> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0</pre>
04282
                     ZPZV<19>, ZPZV<16>, ZPZV<13>, ZPZV<1>, ZPZV<14>, ZPZV<17>, ZPZV<5»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<23, 17> { using ZPZ = aerobus::zpz<23>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     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<18>, ZPZV<18>, ZPZV<2>, ZPZV<1>,
04284
                     ZPZV<18>, ZPZV<3>, ZPZV<16>, ZPZV<21>, ZPZV<0>, ZPZV<11>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<23, 19> { using ZPZ = aerobus::zpz<23>; using type
                     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<5>, ZPZV<5>, ZPZV<18»; }; //</pre>
                     NOLINT
04286
                                   template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type =
                     POLYV<ZPZV<1>, ZPZV<27»; }; // NOLINT
                                   template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
                     POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; }; // NOLINT
04288
                                   template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<27»; }; // NOLINT template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
04289
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<29, 5> { using ZPZ = aerobus::zpz<29>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<27»; }; // NOLINT
                                    template<> struct ConwayPolynomial<29, 6> { using ZPZ = aerobus::zpz<29>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<17>, ZPZV<13>, ZPZV<2»; }; // NOLINT
                                 template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type =
04292
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>; using type = template<> struct ConwayPolynomial<29, 8> { using ZPZ = aerobus::zpz<29>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<2*; };
                    template<> struct ConwayPolynomial<29, 9> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<22>, ZPZV<22>, ZPZV<27»; };
04294
                      // NOLINT
                                  template<> struct ConwayPolynomial<29, 10> { using ZPZ = aerobus::zpz<29>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<8>, ZPZV<17>, ZPZV<2>, ZPZV<22>,
                      ZPZV<2»; }; // NOLINT</pre>
04296
                                 template<> struct ConwayPolynomial<29, 11> { using ZPZ = aerobus::zpz<29>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<8>, ZPZV<27»; }; // NOLINT
  template<> struct ConwayPolynomial<29, 12> { using ZPZ = aerobus::zpz<29>; using type =
04297
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<28>, ZPZV<9>, ZPZV<96>, ZPZV<25>,
                     ZPZV<1>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
04298
                                  template<> struct ConwayPolynomial<29, 13> { using ZPZ = aerobus::zpz<29>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   template<> struct ConwayPolynomial<29, 14> { using ZPZ = aerobus::zpz<29>; using type =
04299
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<3>, ZPZV<14>, ZPZV<10>,
                     ZPZV<21>, ZPZV<18>, ZPZV<27>, ZPZV<5>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::zpz<29>; using type
04300
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<14>, ZPZV<8>, ZPZV<1>, ZPZV<12>, ZPZV<26>, ZPZV<27»; }; // NOLINT
template<> struct ConwayPolynomial<29, 16> { using ZPZ = aerobus::zpz<29>; using type =
04301
```

```
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 =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<27*; }; // NOLINT
template<> struct ConwayPolynomial<29, 18> { using ZPZ = aerobus::zpz<29>; using type =
04303
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<1>,
                             ZPZV<6>, ZPZV<26>, ZPZV<2>, ZPZV<10>, ZPZV<8>, ZPZV<16>, ZPZV<19>, ZPZV<14>, ZPZV<2*; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<29, 19> { using ZPZ = aerobus::zpz<29>; using type =
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                            NOLINT
                                               template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
04305
                            POLYV<ZPZV<1>, ZPZV<28»; }; // NOLINT
04306
                                            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 =
04307
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; // NOLINT
                                               template<> struct ConwayPolynomial<31, 4> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<31, 5> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; }; // NOLINT
                                             template<> struct ConwayPolynomial<31, 6> { using ZPZ = aerobus::zpz<31>; using type =
04310
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<16>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type
04311
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; // NOLINT
04312
                                             template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<12>, ZPZV<24>, ZPZV<3»; };
                            NOLINT
04313
                                             template<> struct ConwayPolynomial<31, 9> { using ZPZ = aerobus::zpz<31>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<20>, ZPZV<29>, ZPZV<28»; };
                             // NOLINT
                                               template<> struct ConwayPolynomial<31, 10> { using ZPZ = aerobus::zpz<31>; using type =
04314
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<26>, ZPZV<13>, ZPZV<13>, ZPZV<13>,
                             ZPZV<3»; }; // NOLINT</pre>
04315
                                                template<> struct ConwayPolynomial<31, 11> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                template<> struct ConwayPolynomial<31, 12> { using ZPZ = aerobus::zpz<31>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<14>, ZPZV<28>, ZPZV<2>, ZPZV<9>, ZPZV<25>, ZPZV<12>, ZPZV<3»; }; // NOLINT
                           template<> struct ConwayPolynomial<31, 13> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04317
                                               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<5>, ZPZV<1>,
                           ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<6>, ZPZV<6>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<31, 15> { 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<3), ZPZV<12>, ZPZV<12>, ZPZV<13>, ZPZV<13>, ZPZV<25>, ZPZV<28»; }; // NOLINT
template<> struct ConwayPolynomial<31, 16> { using ZPZ = aerobus::zpz<31>; using type =
04320
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<28>, ZPZV<24>, ZPZV<26>, ZPZV<28>, ZPZV<11>, ZPZV<19>, ZPZV<27>, ZPZV<3»; }; // NOLINT
                           \label{eq:convayPolynomial} $$ $ \text{TonwayPolynomial} < 1, 17> { using ZPZ = aerobus:: zpz < 31>; using type = POLYV < ZPZV < 1>, ZPZV < 0>, Z
04321
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3, ZPZV<
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2+,
                             ZPZV<2>, ZPZV<7>, ZPZV<12>, ZPZV<11>, ZPZV<25>, ZPZV<25>, ZPZV<10>, ZPZV<6>, ZPZV<3»; }; // NOLINT
                           template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZ
04323
                            NOLINT
                                                template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
04325
                                             template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2»; }; // NOLINT
                                             template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
04326
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT
                                                template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
04327
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<37, 5> { using ZPZ = aerobus::zpz<37>; using type =
04328
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35»; }; // NOLINT template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
04329
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<4>, ZPZV<30>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<37, 7> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>; };
                                             template<> struct ConwayPolynomial<37, 8> { using ZPZ = aerobus::zpz<37>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<20>, ZPZV<27>, ZPZV<27>, ZPZV<1>, ZPZV<2»; };
                           template<> struct ConwayPolynomial<37, 9> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<20>, ZPZV<32>, ZPZV<32>, ZPZV<35»; };</pre>
04332
                            // NOLINT
                          template<> struct ConwayPolynomial<37, 10> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<11>, ZPZV<4>,
                            ZPZV<2»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<37, 11> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
```

```
ZPZV<2>, ZPZV<35»; };
                                                template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<10>, ZPZV<23>, ZPZV<23>,
                             ZPZV<18>, ZPZV<33>, ZPZV<2»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type
04336
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35»; };</pre>
                                                                                                                                                                                                                             // NOLINT
                                                 template<> struct ConwayPolynomial<37, 14> { using ZPZ = aerobus::zpz<37>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<35>, ZPZV<35>, ZPZV<1>,
                             \mbox{ZPZV<32>, ZPZV<16>, ZPZV<1>, ZPZV<9>, ZPZV<2*; }; // NOLINT
                                             template<> struct ConwayPolynomial<37, 15> { using ZPZ = aerobus::zpz<37>; using type =
04338
                            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
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>; // NOLINT template<> struct ConwayPolynomial<37, 18> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<8>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<32>, ZPZV<32>, ZPZV<32>, ZPZV<2>; // NOLINT
                                                template<> struct ConwayPolynomial<37, 19> { using ZPZ = aerobus::zpz<37>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<36>, ZPZV<33>, ZPZV<35»; }; //</pre>
                             NOLINT
                                               template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type =
04342
                            POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                                template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
                            POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6»; }; // NOLINT
 04344
                                               template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<35»; ); // NOLINT
template<> struct ConwayPolynomial<41, 4> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<6»; }; // NOLINT
template<> struct ConwayPolynomial<41, 5> { using ZPZ = aerobus::zpz<41>; using type =
04345
 04346
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<14>, ZPZV<35»; }; // NOLINT
 04347
                                            template<> struct ConwayPolynomial<41, 6> { using ZPZ = aerobus::zpz<41>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<6>, ZPZV<6 , ZPZV<6
04348
                           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
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<35>, ZPZV<35»; };
                              // NOLINT
                            template<> struct ConwayPolynomial<41, 10> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<8>, ZPZV<8>, ZPZV<30>, ZPZV<30>,
04351
                             ZPZV<6»; }; // NOLINT</pre>
 04352
                                             template<> struct ConwayPolynomial<41, 11> { using ZPZ = aerobus::zpz<41>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<20>, ZPZV<35»; }; // NOLINT</pre>
04353
                                              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<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<25, ZPZV<26, ZPZV<26, ZPZV<26, ZPZV<26, ZPZV<26, ZPZV<26, ZPZV<27, ZPZV<28, ZPZV<29, ZPZV<28, ZPZV<29, ZPZV<28, Z
                                               template<> struct ConwayPolynomial<41, 13> { using ZPZ = aerobus::zpz<41>; using type =
 04354
                             \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ 
                            ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<35»; }; // NOLINT
   template<> struct ConwayPolynomial<41, 14> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<12>, ZPZV<15>, ZPZV<4>,
04355
                              ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type =
 04356
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35»; }; // NOLINT
    template<> struct ConwayPolynomial<41, 17> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04357
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<35»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<41, 18> { using ZPZ = aerobus::zpz<41>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<23>, ZPZV<35>, ZPZV<38>, ZPZV<24>, ZPZV<12>, ZPZV<29>, ZPZV<10>, ZPZV<6>, ZPZV<6>, ZPZV<6>; }; // NOLINT
template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
04359
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35»; }; //</pre>
                             NOLINT
                                                 template<> struct ConwayPolynomial<43, 1> { using ZPZ = aerobus::zpz<43>; using type =
 04360
                            POLYV<ZPZV<1>, ZPZV<40»; }; // NOLINT
                                                template<> struct ConwayPolynomial<43, 2> { using ZPZ = aerobus::zpz<43>; using type =
04361
                            POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3»: }; // NOLINT
                                                template<> struct ConwayPolynomial<43, 3> { using ZPZ = aerobus::zpz<43>; using type =
04362
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40»; }; // NOLINT
                                             template<> struct ConwayPolynomial<43, 4> { using ZPZ = aerobus::zpz<43>; using type =
 04363
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<43, 5> { using ZPZ = aerobus::zpz<43>; using type =
04364
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<40»; }; // NOLINT
                                              template<> struct ConwayPolynomial<43, 6> { using ZPZ = aerobus::zpz<43>; using type =
 04365
                           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 =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<42>, ZPZV<7>, ZPZV<40»; };
                            template<> struct ConwayPolynomial<43, 8> { using ZPZ = aerobus::zpz<43>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<39>, ZPZV<20>, ZPZV<24>, ZPZV<3»; }; //</pre>
04367
```

```
template<> struct ConwayPolynomial<43, 9> { using ZPZ = aerobus::zpz<43>; using type :
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<40»; };
                              // NOLINT
                             template<> struct ConwayPolynomial<43, 10> { using ZPZ = aerobus::zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<26>, ZPZV<36>, ZPZV<5>, ZPZV<5>, ZPZV<27>, ZPZV<24>,
04369
                              ZPZV<3»: }: // NOLINT
                                                  template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<7>, ZPZV<40»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<43, 12> { using ZPZ = aerobus::zpz<43>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<27>, ZPZV<16>, ZPZV<17>, ZPZV<6>,
ZPZV<23>, ZPZV<38>, ZPZV<38>, ZPZV<38>; }; // NOLINT
04371
                                                  template<> struct ConwayPolynomial<43, 13> { using ZPZ = aerobus::zpz<43>; using type
04372
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<40»; };</pre>
                                                                                                                                                                                                                                           // NOLINT
                                                  template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type =
04373
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>,
                             ZPZV<37>, ZPZV<18>, ZPZV<19>, ZPZV<19>, ZPZV<39>, ZPZV<37>, ZPZV<37>, ZPZV<38>, ZPZV<39>, ZPZV<39>, ZPZV<39>, ZPZV<39>, ZPZV<30>, ZPZV<3
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZV<3
                             ZPZV<22>, ZPZV<42>, ZPZV<45>, ZPZV<15>, ZPZV<37>, ZPZV<40»; }; // NOLINT
template<> struct ConwayPolynomial<43, 17> { using ZPZ = aerobus::zpz<43>; using type =
04375
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               \texttt{ZPZV} < \texttt{0>, ZPZV} < \texttt{36>, ZPZV} < \texttt{40} \\ \texttt{; }; \ \ // \ \texttt{NOLINT} 
04376
                                                  template<> struct ConwayPolynomial<43, 18> { using ZPZ = aerobus::zpz<43>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<28>, ZPZV<41>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<34>, ZPZV<37>, ZPZV<38>, ZPZV<38<
04377
                                               template<> struct ConwayPolynomial<43, 19> { using ZPZ = aerobus::zpz<43>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              NOLINT
04378
                                                   template<> struct ConwayPolynomial<47, 1> { using ZPZ = aerobus::zpz<47>; using type =
                             POLYV<ZPZV<1>, ZPZV<42»; }; // NOLINT
04379
                                               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 =
04380
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42»; }; // NOLINT template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; }; // NOLINT
                                                template<> struct ConwayPolynomial<47, 5> { using ZPZ = aerobus::zpz<47>; using type =
04382
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<47, 6> { using ZPZ = aerobus::zpz<47>; using type =
04383
                             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
04384
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT
04385
                                                  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»; };
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 // NOLINT
04386
                                               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<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<42»; };
                              // NOLINT
04387
                                                  template<> struct ConwayPolynomial<47, 10> { using ZPZ = aerobus::zpz<47>; using type =
                              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<42>, ZPZV<14>, ZPZV<18>, ZPZV<18>, ZPZV<45>, ZPZV<45>,
                              ZPZV<5»; }; // NOLINT</pre>
04388
                                                 template<> struct ConwayPolynomial<47, 11> { using ZPZ = aerobus::zpz<47>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                   template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type =
                              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<46>, ZPZV<40>, ZPZV<40>, ZPZV<46>,
                              ZPZV<14>, ZPZV<9>, ZPZV<5»; }; // NOLINT</pre>
04390
                                                   template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                  template<> struct ConwayPolynomial<47, 14> { using ZPZ = aerobus::zpz<47>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<30>, ZPZV<30>,
                               ZPZV<17>, ZPZV<24>, ZPZV<9>, ZPZV<32>, ZPZV<5»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<47, 15> { using ZPZ = aerobus::zpz<47>; using type = POLYV<PZVV1>, ZPZV<0>, 
04392
04393
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<42*; }; // NOLINT</pre>
04394
                                               template<> struct ConwayPolynomial<47, 18> { using ZPZ = aerobus::zpz<47>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<41>, ZPZV<42>,
                               \texttt{ZPZV<26>, \ ZPZV<44>, \ ZPZV<22>, \ ZPZV<11>, \ ZPZV<5>, \ ZPZV<45>, \ ZPZV<33>, \ ZPZV<5>; \ \}; \ \ // \ \texttt{NOLINT} } 
                             template<> struct ConwayPolynomial
    tabvels
    httms
    h
04395
                               ZPZV<0>, ZPZV<42»; };</pre>
04396
                                                  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 =
04397
                            POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<53, 3> { using ZPZ = aerobus::zpz<53>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51»; };
                                                                                                                                                                                                                                                                           // NOLINT
04399
                                               template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<38>, ZPZV<38; }; // NOLINT
template<> struct ConwayPolynomial<53, 5> { using ZPZ = aerobus::zpz<53>; using type =
04400
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<51»; };
                                          template<> struct ConwayPolynomial<53, 6> { using ZPZ = aerobus::zpz<53>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<4>, ZPZV<45>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<53, 7> { using ZPZ = aerobus::zpz<53>; using type =
04402
                        template<> struct ConwayPolynomial<53, 8> { using ZPZ = aerobus::zpz<53>; using type =
04403
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<29>, ZPZV<18>, ZPZV<18>, ZPZV<1>, ZPZV<2»; };
                                          template<> struct ConwayPolynomial<53, 9> { using ZPZ = aerobus::zpz<53>; using type
04404
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<5>, ZPZV<51»; };
                         // NOLINT
04405
                                       template<> struct ConwayPolynomial<53, 10> { using ZPZ = aerobus::zpz<53>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<25, ZPZV<25, ZPZV<29>,
                         ZPZV<2»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<53, 11> { using ZPZ = aerobus::zpz<53>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<15>, ZPZV<51»; };</pre>
                                                                                                                      // NOLINT
                        template<> struct ConwayPolynomial<53, 12> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<34>, ZPZV<4>, ZPZV<13>, ZPZV<10>, ZPZV<42>, ZPZV<34>, ZPZV<41>, ZPZV<41>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<53, 13> { using ZPZ = aerobus::zpz<53>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<52>, ZPZV<28>, ZPZV<51»; }; // NOLINT
template<> struct ConwayPolynomial<53, 14> { using ZPZ = aerobus::zpz<53>; using type =
04409
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
                         ZPZV<0>, ZPZV<37>, ZPZV<12>, ZPZV<23>, ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<53, 15> { using ZPZ = aerobus::zpz<53>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51»; }; // NOLINT
template<> struct ConwayPolynomial<53, 17> { using ZPZ = aerobus::zpz<53>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<51»; }; // NOLINT
                                          template<> struct ConwayPolynomial<53, 18> { using ZPZ = aerobus::zpz<53>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<51>,
                         ZPZV<27>, ZPZV<0>, ZPZV<39>, ZPZV<44>, ZPZV<6>, ZPZV<8>, ZPZV<16>, ZPZV<11>, ZPZV<2»; };</pre>
                        template<> struct ConwayPolynomial<53, 19> { using ZPZ = aerobus::zpz<53>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04413
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<51»; }; //</pre>
04414
                                          template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
                        POLYV<ZPZV<1>, ZPZV<57»; }; // NOLINT
04415
                                       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 =
04416
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; }; // NOLINT
                                          template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<2»; }; // NOLINT
04418
                                       template<> struct ConwayPolynomial<59, 5> { using ZPZ = aerobus::zpz<59>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<57»; }; // NOLINT
04419
                                         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
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<57»; }; // NOLINT
04421
                                       template<> struct ConwayPolynomial<59, 8> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<32>, ZPZV<2>, ZPZV<50>, ZPZV<50>, ZPZV<50 //
                         NOLINT
                        template<> struct ConwayPolynomial<59, 9> { using ZPZ = aerobus::zpz<59>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<32>, ZPZV<47>, ZPZV<57»; };</pre>
04422
                        template<> struct ConwayPolynomial<59, 10> { using ZPZ = aerobus::zpz<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28>, ZPZV<25>, ZPZV<4>, ZPZV<39>, ZPZV<15>,
                         ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<59, 11> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                      // NOLINT
                         ZPZV<6>, ZPZV<57»; };</pre>
                                      template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<51>, ZPZV<21>, ZPZV<38>, ZPZV<8>, ZPZV<1>, ZPZV<2»; }; // NOLINT
                                       template<> struct ConwayPolynomial<59, 13> { using ZPZ = aerobus::zpz<59>; using type =
04426
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57»; };</pre>
                                                                                                                                                                                                // NOLINT
                                          template<> struct ConwayPolynomial<59, 14> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<51>, ZPZV<11>,
                         ZPZV<13>, ZPZV<25>, ZPZV<32>, ZPZV<26>, ZPZV<2*; }; // NOLINT
template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type =
04428
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<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
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<57»; }; // NOLINT</pre>
04430
                                       template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59>; using type =
                       template<> struct ConwayPolynomial<59, 19> { using ZPZ = aerobus::zpz<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                         NOLINT
04432
                                        template<> struct ConwayPolynomial<61, 1> { using ZPZ = aerobus::zpz<61>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<59»; };
                                                   template<> struct ConwayPolynomial<61, 2> { using ZPZ = aerobus::zpz<61>; using type =
                              POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
04434
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; }; // NOLINT template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
04435
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2»; }; // NOLINT
04436
                                                    template<> struct ConwayPolynomial<61, 5> { using ZPZ = aerobus::zpz<61>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59»; }; // NOLINT
04437
                                                 template<> struct ConwayPolynomial<61, 6> { using ZPZ = aerobus::zpz<61>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>>, ZPZV<3>, ZPZV<2>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<61, 7> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 
04438
                                                    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<55>, ZPZV<56>, ZPZV<56>, ZPZV<2»; };
04440
                                                template<> struct ConwayPolynomial<61, 9> { using ZPZ = aerobus::zpz<61>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<5
                               // NOLINT
                                                   template<> struct ConwayPolynomial<61, 10> { using ZPZ = aerobus::zpz<61>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<28>, ZPZV<15>, ZPZV<44>, ZPZV<16>, ZPZV<6>,
                               ZPZV<2»; }; // NOLINT</pre>
04442
                                                   template<> struct ConwayPolynomial<61, 11> { using ZPZ = aerobus::zpz<61>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                   template<> struct ConwayPolynomial<61, 12> { using ZPZ = aerobus::zpz<61>; using type =
                              POLYY<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<42>, ZPZV<42>, ZPZV<43>, ZPZV<38>, ZPZV<14>, ZPZV<15>, ZPZV<2»; }; // NOLINT
                                                template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type =
                              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                              template<> struct ConwayPolynomial<61, 14> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<48>, ZPZV<26>, ZPZV<11>,
                                ZPZV<8>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<2»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<61, 15> { using ZPZ = aerobus::zpz<61>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<35>, ZPZV<44>, ZPZV<45>, ZPZV<25>, ZPZV<251>, ZPZV<59»; }; // NOLINT template<> struct ConwayPolynomial<61, 17> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
04447
                                \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0
                                                 template<> struct ConwayPolynomial<61, 18> { using ZPZ = aerobus::zpz<61>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<35>, ZPZV<36>, ZPZV<36>, ZPZV<36>, ZPZV<36>, ZPZV<37>, ZPZV<36>, ZPZV<38
, ZPZV<36>, ZPZV<38
, ZPZV<
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; }; //</pre>
                               NOLINT
04450
                                                 template<> struct ConwayPolynomial<67, 1> { using ZPZ = aerobus::zpz<67>; using type =
                              POLYV<ZPZV<1>, ZPZV<65»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<67, 2> { using ZPZ = aerobus::zpz<67>; using type =
04451
                              POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
04452
                                                   template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65»; }; // NOLINT
04453
                                                 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 =
04454
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<67, 6> { using ZPZ = aerobus::zpz<67>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<63>, ZPZV<49>, ZPZV<55>, ZPZV<2»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<67, 7> { using ZPZ = aerobus::zpz<67>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<65»; };
04457
                                                 template<> struct ConwayPolynomial<67, 8> { using ZPZ = aerobus::zpz<67>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<17>, ZPZV<64>, ZPZV<29; }; //
                              NOLINT
                                                 template<> struct ConwayPolynomial<67, 9> { using ZPZ = aerobus::zpz<67>; using type :
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<49>, ZPZV<55>, ZPZV<65»; };
                               // NOLINT
04459
                              template<> struct ConwayPolynomial<67, 10> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<0>, ZPZV<16>, ZPZV<7>, ZPZV<23>,
                               ZPZV<2»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<67, 11> { using ZPZ = aerobus::zpz<67>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               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<4>, ZPZV<55>, ZPZV<64>,
                               ZPZV<21>, ZPZV<27>, ZPZV<2»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<65»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<67, 14> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<22>, ZPZV<55>, ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<37>, ZPZV<28; }; // NOLINT
                                                    template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<52, ZPZV<41>, ZPZV<20>, ZPZV<20>, ZPZV<65»; }; // NOLINT
                              template<> struct ConwayPolynomial<67, 17> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
```

```
template<> struct ConwayPolynomial<67, 18> { using ZPZ = aerobus::zpz<67>; using type =
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<1>, ZPZV<63>, ZPZV<55>, ZPZV<18>, ZPZV<33>, ZPZV<55>, ZPZV<28>, ZPZV<29>, ZPZV<51>, ZPZV<59>, ZPZV<59>, ZPZV<13>, ZPZV<28; }; // NOLINT
                                          template<> struct ConwayPolynomial<67, 19> { using ZPZ = aerobus::zpz<67>; using type =
                           POLYYCZPZVCO, ZPZVCO, 
                           NOLINT
04468
                                              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 =
04469
                          POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7»; }; // NOLINT
                                            template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
04470
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; }; // NOLINT template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
04471
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, ZPZV<7»; }; // NOLINT
04472
                                             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
                                              template<> struct ConwayPolynomial<71, 6> { using ZPZ = aerobus::zpz<71>; using type =
04473
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<13>, ZPZV<29>, ZPZV<7»; }; // NOLINT
04474
                                            template<> struct ConwayPolynomial<71, 7> { using ZPZ = aerobus::zpz<71>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<64»; }; //
04475
                                            template<> struct ConwayPolynomial<71, 8> { using ZPZ = aerobus::zpz<71>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<22>, ZPZV<19>, ZPZV<7»; }; //
                           NOLINT
04476
                                             template<> struct ConwayPolynomial<71, 9> { using ZPZ = aerobus::zpz<71>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43>, ZPZV<43>, ZPZV<62>, ZPZV<64»; };
                           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>,
04477
                           ZPZV<7»: }: // NOLINT
                           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<0>, ZPZV<12>, ZPZV<28>, ZPZV<29>, ZPZV<55>, ZPZV<21>, ZPZV<58>, ZPZV<23>, ZPZV<7»; }; // NOLINT
                                             template<> struct ConwayPolynomial<71, 13> { using ZPZ = aerobus::zpz<71>; using type
04480
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<27>, ZPZV<64»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<71, 15> { using ZPZ = aerobus::zpz<71>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>,
                           ZPZV<32>, ZPZV<18>, ZPZV<52>, ZPZV<67>, ZPZV<49>, ZPZV<64»; }; // NOLINT
    template<> struct ConwayPolynomial<71, 17> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04482
                            \texttt{ZPZV} < \texttt{O}>, \ \texttt{ZPZV} < \texttt{DVZV} < 
04483
                                           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<4>, ZPZV<4>, ZPZV<64»; }; //</pre>
                           NOLINT
04484
                                              template<> struct ConwavPolynomial<73, 1> { using ZPZ = aerobus::zpz<73>; using type =
                           POLYV<ZPZV<1>, ZPZV<68»; }; // NOLINT
                                              template<> struct ConwayPolynomial<73, 2> { using ZPZ = aerobus::zpz<73>; using type =
                           POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5»; }; // NOLINT
                                            template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
04486
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68»; }; // NOLINT template<> struct ConwayPolynomial<73, 4> { using ZPZ = aerobus::zpz<73>; using type =
04487
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5»; };
                                                                                                                                                                                                                                                                                              // NOLINT
                                           template<> struct ConwayPolynomial<73, 5> { using ZPZ = aerobus::zpz<73>; using type =
04488
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68»; }; // NOLINT
04489
                                              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 =
04490
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<68»; }; // NOLINT
                                              template<> struct ConwayPolynomial<73, 8> { using ZPZ = aerobus::zpz<73>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<53>, ZPZV<39>, ZPZV<18>, ZPZV<5»; }; //
                           NOLINT
04492
                           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<72>, ZPZV<15>, ZPZV<68»; };</pre>
04493
                                              template<> struct ConwayPolynomial<73, 10> { using ZPZ = aerobus::zpz<73>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<23>, ZPZV<33>, ZPZV<32>, ZPZV<69>,
                           ZPZV<5»; }; // NOLINT</pre>
                           \label{eq:convayPolynomial} $$ \ 10^2 = aerobus::zpz<73>; using type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<
04494
                                                                                                                                     // NOLINT
                           ZPZV<5>, ZPZV<68»; };</pre>
                                              template<> struct ConwayPolynomial<73, 12> { using ZPZ = aerobus::zpz<73>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<52>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<46>,
                            ZPZV<29>, ZPZV<25>, ZPZV<5»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<73, 13> { using ZPZ = aerobus::zpz<73>; using type =
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              template<> struct ConwayPolynomial<73, 15> { using ZPZ = aerobus::zpz<73>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           template<> struct ConwayPolynomial<73, 17> { using ZPZ = aerobus::zpz<73>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04498
```

```
template<> struct ConwayPolynomial<73, 19> { using ZPZ = aerobus::zpz<73>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68»; }; //</pre>
                                 NOLINT
 04500
                                                      template<> struct ConwayPolynomial<79, 1> { using ZPZ = aerobus::zpz<79>; using type =
                                POLYV<ZPZV<1>, ZPZV<76»; }; // NOLINT
                                                       template<> struct ConwayPolynomial<79, 2> { using ZPZ = aerobus::zpz<79>; using type =
                                 POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
 04502
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76»; }; // NOLINT template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
04503
                                POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<79, 5> { using ZPZ = aerobus::zpz<79>; using type =
04504
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<76»; }; // NOLINT
 04505
                                                   template<> struct ConwayPolynomial<79, 6> { using ZPZ = aerobus::zpz<79>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<68>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<79, 7> { using ZPZ = aerobus::zpz<79>; using type =
04506
                               POLYV-ZPZV-1>, ZPZV-0>, ZPZV-0>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<5+, ZPZV<5
04508
                                                    template<> struct ConwayPolynomial<79, 9> { using ZPZ = aerobus::zpz<79>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<57>, ZPZV<19>, ZPZV<76»; };
                                 // NOLINT
04509
                                                      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>
 04510
                                                   template<> struct ConwayPolynomial<79, 11> { using ZPZ = aerobus::zpz<79>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<3>, ZPZV<76»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<79, 12> { using ZPZ = aerobus::zpz<79>; using type =
                                POLYYCZPZVC1>, ZPZVC3>, ZPZVC3>, ZPZVC4>, ZPZVC4>, ZPZVC4>, ZPZVC4>, ZPZVC4>, ZPZVC4>, ZPZVC4>, ZPZVC5>, ZPZVC5>, ZPZVC3>; // NOLINT
                                                  template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                     template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type
04513
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 2>, \ \texttt{ZPZV} < 2>, \ \texttt{ZPZV} < 76»; \ \}; \ // \ \texttt{NOLINT} 
                                template<> struct ConwayPolynomial
19> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZ
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<26»; }; //</pre>
                                 NOLINT
                                                     template<> struct ConwayPolynomial<83, 1> { using ZPZ = aerobus::zpz<83>; using type =
                                POLYV<ZPZV<1>, ZPZV<81»; }; // NOLINT
 04516
                                                       template<> struct ConwayPolynomial<83, 2> { using ZPZ = aerobus::zpz<83>; using type =
                                POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; }; // NOLINT
04517
                                                    template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<83, 4> { using ZPZ = aerobus::zpz<83>; using type =
 04518
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2»; }; // NOLINT
                                                       template<> struct ConwayPolynomial<83, 5> { using ZPZ = aerobus::zpz<83>; using type =
 04519
                                04520
                                                     template<> struct ConwayPolynomial<83, 6> { using ZPZ = aerobus::zpz<83>; using type =
                              POLYV<ZPZV1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<12>, ZPZV<12>, ZPZV<15), ZPZV<12>, ZPZV<15), ZPZV<15, ZPZV<15), ZPZV<15, ZPZV<15), ZPZV<15, ZPZV<15), ZPZV<15, ZPZV<15), ZPZV<15, ZPZV<15), ZPZV<15, ZPZV<15, ZPZV<15), ZPZV<15, ZPZV<
04521
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<83, 8> { using ZPZ = aerobus::zpz<83>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<23>, ZPZV<42>, ZPZV<2»; };
                                 NOLINT
04523
                                                    template<> struct ConwayPolynomial<83, 9> { using ZPZ = aerobus::zpz<83>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<24>, ZPZV<18>, ZPZV<81»; };
                                  // NOLINT
                                                      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<5
                                 ZPZV<2»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<81»; }; // NOLINT</pre>
04525
                                                       template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type
                                POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
04527
                                                    template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<81»; }; // NOLINT
  template<> struct ConwayPolynomial<83, 17> { 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<7>, ZPZV<81»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                                      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 =
 04531
                               POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<86»; };
                                                                                                                                                                                                 // NOLINT
                                     template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; }; // NOLINT
                                    template<> struct ConwayPolynomial<89, 5> { using ZPZ = aerobus::zpz<89>; using type =
04534
                     template<> struct ConwayPolynomial<89, 6> { using ZPZ = aerobus::zpz<89>; using type =
04535
                     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
04536
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<86»; }; // NOLINT
04537
                                   template<> struct ConwayPolynomial<89, 8> { using ZPZ = aerobus::zpz<89>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<40>, ZPZV<79>, ZPZV<3»; };
                      NOLINT
                                    template<> struct ConwayPolynomial<89, 9> { using ZPZ = aerobus::zpz<89>; using type
04538
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<1>, ZPZV<6>, ZPZV<86»; };
                       // NOLINT
                                    template<> struct ConwayPolynomial<89, 10> { using ZPZ = aerobus::zpz<89>; using type
04539
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<33>, ZPZV<82>, ZPZV<82>, ZPZV<52>, ZPZV<4>,
                      ZPZV<3»; }; // NOLINT
                                     template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type
04540
                       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>
04541
                                     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
04542
                                     template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type =
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
04543
                                   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<0>, ZPZV<86»; }; // NOLINT template<> struct ConwayPolynomial<89, 19> { using ZPZ = aerobus::zpz<89>; using type
04544
                      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<34>, ZPZV<86»; }; //</pre>
04545
                                     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 =
04546
                     POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5»; }; // NOLINT
04547
                                     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 =
04548
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; }; // NOLINT

template<> struct ConwayPolynomial<97, 5> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<92»; }; // NOLINT
04549
                                     template<> struct ConwayPolynomial<97, 6> { using ZPZ = aerobus::zpz<97>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<92>, ZPZV<58>, ZPZV<88>, ZPZV<5»; }; // NOLINT
04551
                                   template<> struct ConwayPolynomial<97, 7> { using ZPZ = aerobus::zpz<97>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<92»; }; // NOLINT
04552
                                     template<> struct ConwayPolynomial<97, 8> { using ZPZ = aerobus::zpz<97>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>; };
04553
                                     template<> struct ConwayPolynomial<97, 9> { using ZPZ = aerobus::zpz<97>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59>, ZPZV<7>, ZPZV<92»; };
                       // NOLINT
04554
                                     template<> struct ConwayPolynomial<97, 10> { using ZPZ = aerobus::zpz<97>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<20>,
                      ZPZV<5»; }; // NOLINT
                                     template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<5>, ZPZV<92»; }; // NOLINT</pre>
04556
                                     template<> struct ConwayPolynomial<97, 12> { using ZPZ = aerobus::zpz<97>; using type
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<59>, ZPZV<81>, ZPZV<86>, ZPZV<86>, ZPZV<78>, ZPZV<94>, ZPZV<59; // NOLINT
                                     template<> struct ConwayPolynomial<97, 13> { using ZPZ = aerobus::zpz<97>; using type
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                     template<> struct ConwayPolynomials97, 17> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04558
                      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
04559
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<92»; }; //</pre>
04560
                                     template<> struct ConwayPolynomial<101, 1> { using ZPZ = aerobus::zpz<101>; using type =
                     POLYV<ZPZV<1>, ZPZV<99»: }; // NOLINT
                                     template<> struct ConwayPolynomial<101, 2> { using ZPZ = aerobus::zpz<101>; using type =
04561
                      POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
04562
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<99»; ); // NOLINT template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
04563
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<78>, ZPZV<2»; }; // NOLINT
                                     template<> struct ConwayPolynomial<101, 5> { using ZPZ = aerobus::zpz<101>; using type =
04564
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<99»; }; // NOLINT
                                     template<> struct ConwayPolynomial<101, 6> { using ZPZ = aerobus::zpz<101>; using type =
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<90>, ZPZV<20>, ZPZV<67>, ZPZV<2»; }; // NOLINT
04566
                                  template<> struct ConwayPolynomial<101, 7> { using ZPZ = aerobus::zpz<101>; using type =
                    POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
04567
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76>, ZPZV<29>, ZPZV<24>, ZPZV<2»; };
                                  template<> struct ConwayPolynomial<101, 9> { using ZPZ = aerobus::zpz<101>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64>, ZPZV<64>, ZPZV<47>, ZPZV<99»; };
                      // NOLINT
                                     template<> struct ConwayPolynomial<101, 10> { using ZPZ = aerobus::zpz<101>; using type =
04569
                      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 =
04570
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<31>, ZPZV<99»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<101, 12> { using ZPZ = aerobus::zpz<101>; using type
04571
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<79>, ZPZV<64>, ZPZV<39>, ZPZV<38>, ZPZV<48>, ZPZV<84>, ZPZV<21>, ZPZV<22>; }; // NOLINT
                                  template<> struct ConwayPolynomial<101, 13> { using ZPZ = aerobus::zpz<101>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    template<> struct ConwayPolynomial<101, 17> { using ZPZ = aerobus::zpz<101>; using type :
04573
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<99»; };</pre>
                                    template<> struct ConwayPolynomial<101, 19> { using ZPZ = aerobus::zpz<101>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<99»; }; //</pre>
                      NOLINT
04575
                                     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 =
04576
                      POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; }; // NOLINT
04577
                                     template<> struct ConwayPolynomial<103, 3> { using ZPZ = aerobus::zpz<103>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<98»; }; // NOLINT template<> struct ConwayPolynomial<103, 4> { using ZPZ = aerobus::zpz<103>; using type =
04578
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<88>, ZPZV<5»; };
                                                                                                                                                                                                                                      // NOLINT
                                     template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<98»; }; // NOLINT
04580
                                     template<> struct ConwayPolynomial<103, 6> { using ZPZ = aerobus::zpz<103>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<96>, ZPZV<9>, ZPZV<30>, ZPZV<5»; }; // NOLINT
                                    template<> struct ConwayPolynomial<103, 7> { using ZPZ = aerobus::zpz<103>; using type =
04581
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98»; }; // NOLINT
                                     template<> struct ConwayPolynomial<103, 8> { using ZPZ = aerobus::zpz<103>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<70>, ZPZV<71>, ZPZV<49>, ZPZV<5»; }; //
                                    template<> struct ConwayPolynomial<103, 9> { using ZPZ = aerobus::zpz<103>; using type =
04583
                      POLYV-ZPZV-1>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-97>, 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>,
                      ZPZV<5»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<103, 11> { using ZPZ = aerobus::zpz<103>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<103, 12> { using ZPZ = aerobus::zpz<103>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<74>, ZPZV<23>, ZPZV<94>, ZPZV<20>, ZPZV<81>, ZPZV<29>, ZPZV<88>, ZPZV<5»; }; // NOLINT
04587
                                     template<> struct ConwayPolynomial<103, 13> { using ZPZ = aerobus::zpz<103>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      template<> struct ConwayPolynomial<103, 17> { using ZPZ = aerobus::zpz<103>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<102>, ZPZV<8>, ZPZV<98»; }; // NOLINT</pre>
04589
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<3, ZPZV<3,
                      NOLINT
                                     template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
                      POLYV<ZPZV<1>, ZPZV<105»; }; // NOLINT
04591
                                    template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
                      POLYV<ZPZV<1>, ZPZV<103>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<107, 3> { using ZPZ = aerobus::zpz<107>; using type =
04592
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<105»; }; // NOLINT
                                      template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
04593
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<107, 5> { using ZPZ = aerobus::zpz<107>; using type =
04594
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<105»; }; // NOLINT template<> struct ConwayPolynomial<107, 6> { using ZPZ = aerobus::zpz<107>; using type =
04595
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<52>, ZPZV<22>, ZPZV<79>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<105»; };
04597
                                   template<> struct ConwayPolynomial<107, 8> { using ZPZ = aerobus::zpz<107>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<24>, ZPZV<95>, ZPZV<95 }; //
                      NOLINT
                                    template<> struct ConwayPolynomial<107, 9> { using ZPZ = aerobus::zpz<107>; using type =
04598
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<66>, ZPZV<105»; };
04599
                                    template<> struct ConwayPolynomial<107, 10> { using ZPZ = aerobus::zpz<107>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<94>, ZPZV<61>, ZPZV<83>, ZPZV<83>, ZPZV<95>, ZPZV<2»; }; // NOLINT
04600
                                  template<> struct ConwayPolynomial<107, 11> { using ZPZ = aerobus::zpz<107>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<8>, ZPZV<105»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<107, 12> { using ZPZ = aerobus::zpz<107>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<61>, ZPZV<48>, ZPZV<6>, ZPZV<61>, ZPZV<61>, ZPZV<42>, ZPZV<57>, ZPZV<2»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type =
04602
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105»; };</pre>
                                                                                                                                                                                                                                                // NOLINT
                                                template<> struct ConwayPolynomial<107, 17> { using ZPZ = aerobus::zpz<107>; using type :
04603
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 ZPZV<0>,
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105»; }; // NOLINT
template<> struct ConwayPolynomial<107, 19> { using ZPZ = aerobus::zpz<107>; using type =
04604
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  ZPZV<0>.
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<105»; }; //</pre>
                               NOLINT
04605
                                                   template<> struct ConwayPolynomial<109, 1> { using ZPZ = aerobus::zpz<109>; using type =
                              POLYV<ZPZV<1>, ZPZV<103»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<109, 2> { using ZPZ = aerobus::zpz<109>; using type =
04606
                               POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
                              POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<103»; }; // NOLINT template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
 04608
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<109, 5> { using ZPZ = aerobus::zpz<109>; using type =
 04609
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<109, 6> { using ZPZ = aerobus::zpz<109>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<102>, ZPZV<66>, ZPZV<6°, }; // NOLINT
 04611
                                                 template<> struct ConwayPolynomial<109, 7> { using ZPZ = aerobus::zpz<109>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<143»; };
04612
                                                 template<> struct ConwayPolynomial<109, 8> { using ZPZ = aerobus::zpz<109>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<34>, ZPZV<86>, ZPZV<6»; }; //
                              NOLINT
                                                   template<> struct ConwayPolynomial<109, 9> { using ZPZ = aerobus::zpz<109>; using type =
04613
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<93>, ZPZV<87>, ZPZV<103»; };
                                // NOLINT
04614
                                                  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>
                                                   template<> struct ConwayPolynomial<109, 11> { using ZPZ = aerobus::zpz<109>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<11>, ZPZV<103»; }; // NOLINT
                              \label{eq:convergence} template<> struct ConwayPolynomial<109, 12> \{ using ZPZ = aerobus::zpz<109>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<50>, ZPZV<53>, ZPZV<37>, ZPZV<8>, ZPZV<65>,
04616
                               ZPZV<103>, ZPZV<28>, ZPZV<6»; }; // NOLINT</pre>
                                                   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<1>, ZPZV<103»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<109, 19> { using ZPZ = aerobus::zpz<109>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<103»; }; //</pre>
                               NOLINT
04620
                                                  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 =
                              POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<110»; }; // NOLINT template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
04623
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<62>, ZPZV<62, ZPZV<62, ZPZV<62 , ZPZV×62 , ZPZV×6
 04624
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110»; }; // NOLINT
04625
                                                 template<> struct ConwayPolynomial<113, 6> { using ZPZ = aerobus::zpz<113>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<59>, ZPZV<30>, ZPZV<71>, ZPZV<3»; }; // NOLINT
 04626
                                                   template<> struct ConwayPolynomial<113, 7> { using ZPZ = aerobus::zpz<113>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZP
04627
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<98>, ZPZV<38>, ZPZV<28>, ZPZV<3»; };
04628
                                               template<> struct ConwayPolynomial<113, 9> { using ZPZ = aerobus::zpz<113>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<87>, ZPZV<87>, ZPZV<71>, ZPZV<110»; };
                               // NOLINT
                                                   template<> struct ConwayPolynomial<113, 10> { using ZPZ = aerobus::zpz<113>; using type =
04629
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<56>,
                               ZPZV<3»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<3>, ZPZV<110»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<113, 12> { using ZPZ = aerobus::zpz<113>; using type :
04631
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<23>, ZPZV<62>, ZPZV<4>, ZPZV<98>, ZPZV<56>, ZPZV<10>, ZPZV<27>, ZPZV<3»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<113, 13> { using ZPZ = aerobus::zpz<113>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                template<> struct ConwayPolynomial<113, 17> { using ZPZ = aerobus::zpz<113>; using type =
04633
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110»; }; // NOLINT
template<> struct ConwayPolynomial<113, 19> { using ZPZ = aerobus::zpz<113>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<110»; }; //</pre>
                            NOLINT
                                              template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
                           POLYV<ZPZV<1>, ZPZV<124»; }; // NOLINT
                                           template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
                          POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
04637
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124»; }; // NOLINT
                           04638
 04639
                                           template<> struct ConwayPolynomial<127, 5> { using ZPZ = aerobus::zpz<127>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<5, 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 template<> struct ConwayPolynomial<127, 7> { using ZPZ = aerobus::zpz<127>; using type =
04640
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<124»; }; //
                                            template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<104>, ZPZV<55>, ZPZV<8>, ZPZV<8»; }; //
                           NOLINT
                                            template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type =
04643
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<12
                            }; // NOLINT
                                           template<> struct ConwayPolynomial<127, 10> { using ZPZ = aerobus::zpz<127>; using type :
04644
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<64>, ZPZV<95>, ZPZV<60>, ZPZV<4>,
                            ZPZV<3»; }; // NOLINT</pre>
04645
                                            template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<11>, ZPZV<124»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<127, 12> { using ZPZ = aerobus::zpz<127>; using type
04646
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<119>, ZPZV<25>, ZPZV<33>, ZPZV<97>, ZPZV<15>,
                            ZPZV<99>, ZPZV<8>, ZPZV<3»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type =
04647
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              template<> struct ConwayPolynomial<127, 17> { using ZPZ = aerobus::zpz<127>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                         // NOLINT
                            template<> struct ConwayPolynomial<127, 19> { using ZPZ = aerobus::zpz<127>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04649
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>, ZPZV<124»; }; //</pre>
                            NOLINT
04650
                                              template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
                           POLYV<ZPZV<1>, ZPZV<129»; }; // NOLINT
                                            template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
04651
                           POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2»; }; // NOLINT
                                              template<> struct ConwayPolynomial<131, 3> { using ZPZ = aerobus::zpz<131>; using type =
 04652
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<129»; }; // NOLINT template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type =
                           template<> struct ConwayPolynomial<131, 5> { using ZPZ = aerobus::zpz<131>; using type =
 04654
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<129»; }; // NOLINT
                                              template<> struct ConwayPolynomial<131, 6> { using ZPZ = aerobus::zpz<131>; using type =
04655
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<4>, ZPZV<22>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<131, 7> { using ZPZ = aerobus::zpz<131>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<1299; }; // NOLINT template<> struct ConwayPolynomial<131, 8> { using ZPZ = aerobus::zpz<131>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<72>, ZPZV<116>, ZPZV<104>, ZPZV<2»; }; //
 04657
                           NOLINT
04658
                                             template<> struct ConwayPolynomial<131, 9> { using ZPZ = aerobus::zpz<131>; using type =
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<19>, ZPZV<129»; };
                           template<> struct ConwayPolynomial<131, 10> { using ZPZ = aerobus::zpz<131>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<124>, ZPZV<97>, ZPZV<9>, ZPZV<126>, ZPZV<44>,
04659
                            ZPZV<2»: }: // NOLINT</pre>
                                            template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type =
04660
                            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 =
04661
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<122>, ZPZV<40>, ZPZV<83>, ZPZV<125>,
                            ZPZV<28>, ZPZV<103>, ZPZV<2»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type =
04662
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<129»; };</pre>
                                                                                                                                                                                                                         // NOLINT
04663
                                            template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<129»; }; // NOLINT
template<> struct ConwayPolynomial<131, 19> { using ZPZ = aerobus::zpz<131>; using type :
04664
                            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<9>, ZPZV<9>, ZPZV<129»; }; //</pre>
                            NOLINT
 04665
                                            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»; };
                                           template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
 04668
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<3>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
04669
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<134»; }; // NOLINT
 04670
                                           template<> struct ConwayPolynomial<137, 6> { using ZPZ = aerobus::zpz<137>; using type =
                         04671
                                         template<> struct ConwayPolynomial<137, 7> { using ZPZ = aerobus::zpz<137>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>; ZPZV<1>; zPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 8> { using ZPZ = aerobus::zpz<137>; using type =
04672
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105>, ZPZV<21>, ZPZV<34>, ZPZV<33+, ZPZV<3»; };
 04673
                                        template<> struct ConwayPolynomial<137, 9> { using ZPZ = aerobus::zpz<137>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<80>, ZPZV<122>, ZPZV<134»;
                          }; // NOLINT
04674
                                           template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type :
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<20>, ZPZV<67>, ZPZV<93>, ZPZV<119>,
                          ZPZV<3»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<137, 11> { using ZPZ = aerobus::zpz<137>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<1>, ZPZV<134»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<137, 12> { using ZPZ = aerobus::zpz<137>; using type =
04676
                         POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<61>, ZPZV<40>, ZPZV<40>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<30>, ZPZV<
                                         template<> struct ConwayPolynomial<137, 13> { using ZPZ = aerobus::zpz<137>; using type =
 04677
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<134»; }; // NOLINT
template<> struct ConwayPolynomial<137, 17> { using ZPZ = aerobus::zpz<137>; using type =
04678
                         POLYV<PPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<4>, ZPZV<134»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<137, 19> { using ZPZ = aerobus::zpz<137>; using type =
04679
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<134»; }; //</pre>
                          NOLINT
04680
                                          template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
                         POLYV<ZPZV<1>, ZPZV<137»; }; // NOLINT
                                            template<> struct ConwayPolynomial<139, 2> { using ZPZ = aerobus::zpz<139>; using type =
                          POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2»; }; // NOLINT
04682
                                         template<> struct ConwayPolynomial<139, 3> { using ZPZ = aerobus::zpz<139>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<137»; }; // NOLINT template<> struct ConwayPolynomial<139, 4> { using ZPZ = aerobus::zpz<139>; using type =
04683
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2»; };
                                                                                                                                                                                                                                                                      // NOLINT
                                            template<> struct ConwayPolynomial<139, 5> { using ZPZ = aerobus::zpz<139>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<137»; }; // NOLINT
 04685
                                         template<> struct ConwayPolynomial<139, 6> { using ZPZ = aerobus::zpz<139>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<46>, ZPZV<10>, ZPZV<118>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<139, 7> { using ZPZ = aerobus::zpz<139>; using type =
04686
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<137»; }; // NOLINT
                                           template<> struct ConwayPolynomial<139, 8> { using ZPZ = aerobus::zpz<139>; using type
04687
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>, ZPZV<36>, ZPZV<21>, ZPZV<22), };
                          NOLINT
04688
                                         template<> struct ConwayPolynomial<139, 9> { using ZPZ = aerobus::zpz<139>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<70>, ZPZV<87>, ZPZV<87
, ZPZV<87
                          // NOLINT
                                           template<> struct ConwayPolynomial<139, 10> { using ZPZ = aerobus::zpz<139>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<110>, ZPZV<48>, ZPZV<130>, ZPZV<66>,
                           ZPZV<106>, ZPZV<2»; }; // NOLINT</pre>
04690
                                            template<> struct ConwayPolynomial<139, 11> { using ZPZ = aerobus::zpz<139>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<137»; }; // NOLINT
04691
                                           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<141>, ZPZV<77>, ZPZV<106>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
04692
                                        template<> struct ConwayPolynomial<139, 13> { using ZPZ = aerobus::zpz<139>; using type =
                          \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                          ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<137»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type =
04693
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137»; };</pre>
04694
                                        template<> struct ConwayPolynomial<139, 19> { using ZPZ = aerobus::zpz<139>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          NOLINT
                                           template<> struct ConwayPolynomial<149, 1> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<147»; }; // NOLINT
                                         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 =
04697
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
 04698
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<2»; }; // NOLINT
 04699
                                         template<> struct ConwayPolynomial<149, 5> { using ZPZ = aerobus::zpz<149>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 6> { using ZPZ = aerobus::zpz<149>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<2>; }; // NOLINT
 04700
```

```
template<> struct ConwayPolynomial<149, 7> { using ZPZ = aerobus::zpz<149>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<19>, ZPZV<147»; }; // NOLINT
                                        template<> struct ConwayPolynomial<149, 8> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<140>, ZPZV<25>, ZPZV<123>, ZPZV<22»; }; //
                         NOLINT
                                         template<> struct ConwayPolynomial<149, 9> { using ZPZ = aerobus::zpz<149>; using type =
04703
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<146>, ZPZV<20), ZPZV<147»;
                         }; // NOLINT
04704
                                         template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<74>, ZPZV<42>, ZPZV<148>, ZPZV<143>, ZPZV<51>,
                         ZPZV<2»: }: // NOLINT</pre>
04705
                                         template<> struct ConwayPolynomial<149, 11> { using ZPZ = aerobus::zpz<149>; using type :
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type :
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<91>, ZPZV<52>, ZPZV<9>, ZPZV<104>, ZPZV<110>, ZPZV<2»; }; // NOLINT
04707
                                         template<> struct ConwayPolynomial<149, 13> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<147»; };</pre>
                                                                                                                                                                                                           // NOLINT
                                         template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<147»; }; // NOLINT
    template<> struct ConwayPolynomial<149, 19> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, 
04709
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5</pre>, //
04710
                                         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 =
04711
                         POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6»; }; // NOLINT
04712
                                           template<> struct ConwayPolynomial<151, 3> { using ZPZ = aerobus::zpz<151>; using type =
                         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<145»; }; // NOLINT template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
04713
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<151, 5> { using ZPZ = aerobus::zpz<151>; using type =
04714
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0), ZPZV<11>, ZPZV<145»; }; // NOLINT template<> struct ConwayPolynomial<151, 6> { using ZPZ = aerobus::zpz<151>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<125>, ZPZV<18>, ZPZV<15>, ZPZV<6»; }; // NOLINT
                                        template<> struct ConwayPolynomial<151, 7> { using ZPZ = aerobus::zpz<151>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<145»; }; // NOLINT
                                         template<> struct ConwayPolynomial<151, 8> { using ZPZ = aerobus::zpz<151>; using type =
04717
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, 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<6>, ZPZV<6>, ZPZV<126>, ZPZV<126>, ZPZV<145»;
                         }; // 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<2PZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<1>, ZPZV<145»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<151, 12> { using ZPZ = aerobus::zpz<151>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<121>, ZPZV<101>, ZPZV<101>, ZPZV<6>, ZPZV<77>,
04721
                         ZPZV<107>, ZPZV<147>, ZPZV<6»; }; // NOLINT
    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>
04723
                                           template<> struct ConwayPolynomial<151,
                                                                                                                                                                                                                  17> { using ZPZ = aerobus::zpz<151>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0</pre>
                         NOLINT
04725
                                          template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
                         POLYV<ZPZV<1>, ZPZV<152»; }; // NOLINT
                                         template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
                         POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5»; }; // NOLINT
                                          template<> struct ConwayPolynomial<157, 3> { using ZPZ = aerobus::zpz<157>; using type =
04727
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<152»; }; // NOLINT template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
04728
                         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<11>, ZPZY<136>, ZPZY<5; }; // NOLINT
template<> struct ConwayPolynomial<157, 5> { using ZPZ = aerobus::zpz<157>; using type =
04729
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<152»; }; // NOLINT
                                         template<> struct ConwayPolynomial<157, 6> { using ZPZ = aerobus::zpz<157>; using type =
04730
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<130>, ZPZV<43>, ZPZV<144>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<157, 7> { using ZPZ = aerobus::zpz<157>; using type =
04731
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<15, zPZV<1
04732
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<97>, ZPZV<40>, ZPZV<153>, ZPZV<5»; };
04733
                                         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
04734
                                       template<> struct ConwayPolynomial<157, 10> { using ZPZ = aerobus::zpz<157>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<61>, ZPZV<22>, ZPZV<124>, ZPZV<61>, ZPZV<93>,
                      ZPZV<5»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<157, 11> { using ZPZ = aerobus::zpz<157>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<29>, ZPZV<152»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<157, 12> { using ZPZ = aerobus::zpz<157>; using type =
04736
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1110>, 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 =
04737
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<156>, ZPZV<9>, ZPZV<152»; // NOLINT template<> struct ConwayPolynomial<157, 17> { using ZPZ = aerobus::zpz<157>; using type =
04738
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<152»; }; // NOLINT</pre>
                                   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>,
                                                                                                                                                                                                                                                                                                                                                           7.P7.V<0>.
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<152»; }; //</pre>
                      NOLINT
                                     template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
                      POLYV<ZPZV<1>, ZPZV<161»; }; // NOLINT
                                      template<> struct ConwayPolynomial<163, 2> { using ZPZ = aerobus::zpz<163>; using type =
                      POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
04742
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<161»; }; // NOLINT
template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
04743
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2»; }; // NOLINT
04744
                                    template<> struct ConwayPolynomial<163, 5> { using ZPZ = aerobus::zpz<163>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<161»; }; // NOLINT
04745
                                      template<> struct ConwayPolynomial<163, 6> { using ZPZ = aerobus::zpz<163>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<25>, ZPZV<156>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<163, 7> { using ZPZ = aerobus::zpz<163>; using type =
04746
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<161»; };
                                     template<> struct ConwayPolynomial<163, 8> { using ZPZ = aerobus::zpz<163>; using type
04747
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<132>, ZPZV<83>, ZPZV<6>, ZPZV<6>, ZPZV<2»; };
                      NOLINT
                                     template<> struct ConwayPolynomial<163, 9> { using ZPZ = aerobus::zpz<163>; using type =
04748
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<15>, ZPZV<162>, ZPZV<127>, ZPZV<161»;
04749
                                       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>
                      \label{eq:convayPolynomial} $$ $$ template<> struct ConwayPolynomial<163, 11> { using ZPZ = aerobus::zpz<163>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04750
                      ZPZV<11>, ZPZV<161»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<112>, ZPZV<31>, ZPZV<38>, ZPZV<103>,
                      ZPZV<10>, ZPZV<69>, ZPZV<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<71>, ZPZV<161»; }; // NOLINT</pre>
                      template<> struct ConwayPolynomial<163, 19> { using ZPZ = aerobus::zpz<163>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
04754
                       ZPZV<0>, ZPZV<6</pre>
                                     template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
                      POLYV<ZPZV<1>, ZPZV<162»; }; // NOLINT
04756
                                      template<> struct ConwayPolynomial<167, 2> { using ZPZ = aerobus::zpz<167>; using type =
                     POLYV<ZPZV<1>, ZPZV<166>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 3> { using ZPZ = aerobus::zpz<167>; using type =
04757
                      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 =
04759
                     04760
                                    template<> struct ConwayPolynomial<167, 6> { using ZPZ = aerobus::zpz<167>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<75>, ZPZV<38>, ZPZV<2>, ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<167, 7> { using ZPZ = aerobus::zpz<167>; using type
04761
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162»; }; // NOLINT template<> struct ConwayPolynomial<167, 8> { using ZPZ = aerobus::zpz<167>; using type =
04762
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<149>, ZPZV<56>, ZPZV<113>, ZPZV<5»; }; //
                      NOLINT
                                    template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type =
04763
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<165>, ZPZV<162»;
                      }; // NOLINT
04764
                                      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>,
                      ZPZV<148>, ZPZV<5»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type :
04765
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<24>, ZPZV<162»; }; // NOLINT</pre>
04766
                                     template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<142>, ZPZV<10>, ZPZV<142>, ZPZV<1
                      ZPZV<140>, ZPZV<41>, ZPZV<57>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 13> { using ZPZ = aerobus::zpz<167>; using type =
04767
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162»; }; // NOLINT
template<> struct ConwayPolynomial<167, 17> { using ZPZ = aerobus::zpz<167>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<32>, ZPZV<162»; }; // NOLINT
template<> struct ConwayPolynomial<167, 19> { using ZPZ = aerobus::zpz<167>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<162»; }; //</pre>
                                                     template<> struct ConwayPolynomial<173, 1> { using ZPZ = aerobus::zpz<173>; using type =
                                POLYV<ZPZV<1>, ZPZV<171»; // NOLINT
                                                     template<> struct ConwayPolynomial<173, 2> { using ZPZ = aerobus::zpz<173>; using type =
                                POLYV<ZPZV<1>, ZPZV<169>, ZPZV<2»; }; // NOLINT
                                                        template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
 04772
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
 04773
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<102>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<173, 5> { using ZPZ = aerobus::zpz<173>; using type =
04774
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, 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<27>, ZPZV<134>, ZPZV<107>, ZPZV<2»; }; // NOLINI
                                                    template<> struct ConwayPolynomial<173, 7> { using ZPZ = aerobus::zpz<173>; using type =
 04776
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 8> { using ZPZ = aerobus::zpz<173>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<125>, ZPZV<158>, ZPZV<27>, ZPZV<27>, ZPZV<2»; }; //
                                                   template<> struct ConwayPolynomial<173, 9> { using ZPZ = aerobus::zpz<173>; using type =
04778
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<104>, ZPZV<171»;
                                  }; // NOLINT
04779
                                                      template<> struct ConwayPolynomial<173, 10> { using ZPZ = aerobus::zpz<173>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<156>, ZPZV<164>, ZPZV<48>, ZPZV<106>,
                                  ZPZV<58>, ZPZV<2»; };</pre>
                                                                                                                                                              // NOLINT
                                                       template<> struct ConwayPolynomial<173, 11> { using ZPZ = aerobus::zpz<173>; using type =
04780
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<12>, ZPZV<171»; }; // NOLINT
                                                       template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type =
04781
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<64>, ZPZV<46>, ZPZV<166>, ZPZV<0>, ZPZV<0>, ZPZV<159>, ZPZV<22>, ZPZV<22»; }; // NOLINT
                                                       template<> struct ConwayPolynomial<173, 13> { using ZPZ = aerobus::zpz<173>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type =
04783
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<1>; // NOLINT 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>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171»; }; //</pre>
                                  NOLINT
                                                      template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
04785
                                 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
 04787
                                                     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 =
04788
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<20; ZPZV<109>, ZPZV<2»; }; // NOLINT 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
 04790
                                                       template<> struct ConwayPolynomial<179, 6> { using ZPZ = aerobus::zpz<179>; using type =
                                  \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 7>, \ \texttt{ZPZV} < 91>, \ \texttt{ZPZV} < 55>, \ \texttt{ZPZV} < 109>, \ \texttt{ZPZV} < 2»; \ \}; \ \ // \ \ \texttt{NOLINT} 
                                template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 
04791
                                                     template<> struct ConwayPolynomial<179, 8> { using ZPZ = aerobus::zpz<179>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<144>, ZPZV<73>, ZPZV<2»; };
04793
                                                   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<40>, ZPZV<40>, ZPZV<40>, ZPZV<64>, ZPZV<64 , ZPZV<6
                                   // NOLINT
                                                     template<> struct ConwayPolynomial<179, 10> { using ZPZ = aerobus::zpz<179>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<71>, ZPZV<75>, ZPZV<45>, ZPZV<89>, ZPZV<8
                                   ZPZV<2»; }; // NOLINT</pre>
04795
                                                   template<> struct ConwayPolynomial<179, 11> { using ZPZ = aerobus::zpz<179>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                       template<> struct ConwayPolynomial<179, 12> { using ZPZ = aerobus::zpz<179>; using type
04796
                                 POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<103>, ZPZV<83>, ZPZV<43>, ZPZV<76>, ZPZV<8>, ZPZV<177>, ZPZV<1>, ZPZV<2»; }; // NOLINT
04797
                                                    template<> struct ConwayPolynomial<179, 13> { using ZPZ = aerobus::zpz<179>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<177»; }; // NOLINT template<> struct ConwayPolynomial<179, 17> { using ZPZ = aerobus::zpz<179>; using type =
04798
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<477; }; // NOLINT
template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177»; }; //</pre>
                                  NOT.TNT
```

```
template<> struct ConwayPolynomial<181, 1> { using ZPZ = aerobus::zpz<181>; using type =
                    POLYV<ZPZV<1>, ZPZV<179»; }; // NOLINT
04801
                                 template<> struct ConwayPolynomial<181, 2> { using ZPZ = aerobus::zpz<181>; using type =
                    POLYV<ZPZV<1>, ZPZV<177>, ZPZV<2»; }; // NOLINT
04802
                                  template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
                    POLYV<ZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<179»; }; // NOLINT template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<105>, ZPZV<2»; }; // NOLINT
04804
                                 template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<179»; }; // NOLINT
04805
                                 \texttt{template<>} \texttt{struct ConwayPolynomial<181, 6> \{ \texttt{using ZPZ = aerobus::zpz<181>; using type = 1.5 \}}
                    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
04806
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<179»; }; //
04807
                                template<> struct ConwayPolynomial<181, 8> { using ZPZ = aerobus::zpz<181>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<108>, ZPZV<22>, ZPZV<149>, ZPZV<2*; };
                     NOLTNT
                    template<> struct ConwayPolynomial<181, 9> { using ZPZ = aerobus::zpz<181>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<107>, ZPZV<168>, ZPZV<179»;
04808
                    }; // NoLINT
template<> struct ConwayPolynomial<181, 10> { using ZPZ = aerobus::zpz<181>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<154>, ZPZV<104>, ZPZV<94>, ZPZV<57>, ZPZV<88>,
                                 template<> struct ConwayPolynomial<181, 11> { using ZPZ = aerobus::zpz<181>; using type =
04810
                     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 =
04811
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<141>, ZPZV<45>, ZPZV<122>,
                    ZPZV<175>, ZPZV<12>, ZPZV<10>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<181, 13> { using ZPZ = aerobus::zpz<181>; using type =
04812
                     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 =
04813
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<179»; }; // NOLINT
template<> struct ConwayPolynomial<181, 19> { using ZPZ = aerobus::zpz<181>; using type =
04814
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36</pre>, //
                                   template<> struct ConwayPolynomial<191, 1> { using ZPZ = aerobus::zpz<191>; using type =
                    POLYV<ZPZV<1>, ZPZV<172»; }; // NOLINT
                                   template<> struct ConwayPolynomial<191, 2> { using ZPZ = aerobus::zpz<191>; using type =
04816
                    POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19»; }; // NOLINT
                                   template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
04817
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172»; }; // NOLINT
04818
                                   template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; }; // NOLINT template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
04819
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<172»; }; // NOLINT
                                   template<> struct ConwayPolynomial<191, 6> { using ZPZ = aerobus::zpz<191>; using type =
04820
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<110>, ZPZV<10>, ZPZV<19»; }; // NOLINT
                                   template<> struct ConwayPolynomial<191, 7> { using ZPZ = aerobus::zpz<191>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<14>, ZPZV<172»; };
04822
                                 template<> struct ConwayPolynomial<191, 8> { using ZPZ = aerobus::zpz<191>; using type =
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<164>, ZPZV<139>, ZPZV<171>, ZPZV<19»; }; //
                     NOLINT
                                   template<> struct ConwayPolynomial<191, 9> { using ZPZ = aerobus::zpz<191>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<62>, ZPZV<124>, ZPZV<172»;
                     }; // NOLINT
04824
                                   template<> struct ConwayPolynomial<191, 10> { using ZPZ = aerobus::zpz<191>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<113>, ZPZV<47>, ZPZV<173>, ZPZV<74>, ZPZV<156>, ZPZV<19»; }; // NOLINT
                                   template<> struct ConwayPolynomial<191, 11> { using ZPZ = aerobus::zpz<191>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04826
                                template<> struct ConwayPolynomial<191, 12> { using ZPZ = aerobus::zpz<191>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<76>, ZPZV<168>, ZPZV<25>, ZPZV<49>, ZPZV<90>,
                     ZPZV<7>, ZPZV<151>, ZPZV<19»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<191, 13> { using ZPZ = aerobus::zpz<191>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<172»; }; // NOLINT</pre>
04828
                                template<> struct ConwayPolynomial<191, 17> { using ZPZ = aerobus::zpz<191>; using type =
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>; ZPZV<172»; }; // NOLINT template<> struct ConwayPolynomial<191, 19> { using ZPZ = aerobus::zpz<191>; using type =
                                   template<> struct ConwayPolynomial<191,
04829
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<190>, ZPZV<2>, ZPZV<172»; }; //</pre>
04830
                                   template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
                    POLYV<7P7V<1>. 7P7V<188»: }: // NOLINT
                                   template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
04831
                    POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5»; }; // NOLINT
                                   template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; };
                                                                                                                                                                                            // NOLINT
                                template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
04833
                  POLYV<2PZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<64, ZPZV<148>, ZPZV<5; }; // NOLINT template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
04834
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<188»; };
                      template<> struct ConwayPolynomial<193, 6> { using ZPZ = aerobus::zpz<193>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<8>, ZPZV<172>, ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<193, 7> { using ZPZ = aerobus::zpz<193>; using type =
04836
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<88, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 8> { using ZPZ = aerobus::zpz<193>; using type =
04837
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<145>, ZPZV<34>, ZPZV<154>, ZPZV<5»; }; //
04838
                                      template<> struct ConwayPolynomial<193, 9> { using ZPZ = aerobus::zpz<193>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<168>, ZPZV<27>, ZPZV<188»;
                        }; // NOLINT
04839
                                       template<> struct ConwayPolynomial<193, 10> { using ZPZ = aerobus::zpz<193>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<51>, ZPZV<77>, ZPZV<78>, ZPZV<89>,
                        ZPZV<5»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; }; // NOLINT
                                      template<> struct ConwayPolynomial<193, 12> { using ZPZ = aerobus::zpz<193>; using type =
04841
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<155>, ZPZV<52>, ZPZV<135>, ZPZV<155>,
                        ZPZV<90>, ZPZV<46>, ZPZV<28>, ZPZV<5»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<193, 13> { using ZPZ = aerobus::zpz<193>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<188»; }; // NOLINT
  template<> struct ConwayPolynomial<193, 17> { using ZPZ = aerobus::zpz<193>; using type :
04843
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<188»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<193,
                                                                                                                                                                                                 19> { using ZPZ = aerobus::zpz<193>; using type =
04844
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<188»; }; //</pre>
                        NOLINT
                                      template<> struct ConwayPolynomial<197, 1> { using ZPZ = aerobus::zpz<197>; using type =
04845
                       POLYV<ZPZV<1>, ZPZV<195»; }; // NOLINT
                                       template<> struct ConwayPolynomial<197, 2> { using ZPZ = aerobus::zpz<197>; using type =
                       POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2»; }; // NOLINT
 04847
                                       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 =
04848
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<124>, ZPZV<2»; }; // NOLINT
 04849
                                        template<> struct ConwayPolynomial<197, 5> { using ZPZ = aerobus::zpz<197>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195»; }; // NOLINT
                      template<> struct ConwayPolynomial<197, 6> { using ZPZ = aerobus::zpz<197>; using type =
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 =
04850
 04851
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<195»; };
                                       template<> struct ConwayPolynomial<197, 8> { using ZPZ = aerobus::zpz<197>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<176>, ZPZV<96>, ZPZV<29>, ZPŽV<2»; };
04853
                                     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<1>, ZPZV<13>, ZPZV<127>, ZPZV<8>, ZPZV<195»;
                        }; // NOLINT
04854
                                        template<> struct ConwayPolynomial<197, 10> { using ZPZ = aerobus::zpz<197>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<137>, ZPZV<8>, ZPZV<73>, ZPZV<42>,
                        ZPZV<2»; }; // NOLINT</pre>
04855
                                       template<> struct ConwayPolynomial<197, 11> { using ZPZ = aerobus::zpz<197>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<14>, ZPZV<195»; }; // NOLINT
    template<> struct ConwayPolynomial<197, 12> { using ZPZ = aerobus::zpz<197>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<168>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>,
                        \text{ZPZV} < 90 >, \text{ZPZV} < 163 >, \text{ZPZV} < 2 >; }; // NOLINT
                                        template<> struct ConwayPolynomial<197, 13> { using ZPZ = aerobus::zpz<197>; using type :
04857
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       template<> struct ConwayPolynomial<197,
                                                                                                                                                                                                 17> { using ZPZ = aerobus::zpz<197>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0 , ZPZ
04859
                        NOLINT
04860
                                        template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
                       POLYV<ZPZV<1>, ZPZV<196»; }; // NOLINT
 04861
                                     template<> struct ConwayPolynomial<199, 2> { using ZPZ = aerobus::zpz<199>; using type =
                       POLYV<ZPZV<1>, ZPZV<193>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<199, 3> { using ZPZ = aerobus::zpz<199>; using type =
 04862
                       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; }; // NOLINT template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; }; // NOLINT
 04864
                                       template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
                                      template<> struct ConwayPolynomial<199, 6> { using ZPZ = aerobus::zpz<199>; using type =
04865
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<90>, ZPZV<58>, ZPZV<79>, ZPZV<3»; }; // NOLINT
 04866
                                       template<> struct ConwayPolynomial<199,
                                                                                                                                                                                                 7> { using ZPZ = aerobus::zpz<199>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3, ZPZV<3,
 04867
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<160>, ZPZV<23>, ZPZV<159>, ZPZV<3»; }; //
                       NOLINT
 04868
                                     template<> struct ConwayPolynomial<199, 9> { using ZPZ = aerobus::zpz<199>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<177>, ZPZV<141>, ZPZV<196»;
                         }; // NOLINT
                                         template<> struct ConwayPolynomial<199, 10> { using ZPZ = aerobus::zpz<199>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<158>, ZPZV<31>, ZPZV<54>, ZPZV<9>,
                         ZPZV<3»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<199, 11> { using ZPZ = aerobus::zpz<199>; using type =
04870
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<1>, ZPZV<196»; };</pre>
                                                                                                                              // NOLINT
                                        template<> struct ConwayPolynomial<199, 12> { using ZPZ = aerobus::zpz<199>; using type =
04871
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<192>, ZPZV<192>, ZPZV<197>, ZPZV<138>, ZPZV<69>, ZPZV<57>, ZPZV<151>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<199, 13> { using ZPZ = aerobus::zpz<199>; using type =
04872
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<196»; };</pre>
                                                                                                                                                                                                           // NOLINT
                                        template<> struct ConwayPolynomial<199, 17> { using ZPZ = aerobus::zpz<199>; using type
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04874
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<196»; }; //</pre>
04875
                                          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 =
04876
                        POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<209»; }; // NOLINT
                                         template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
04878
                         template<> struct ConwayPolynomial<211, 5> { using ZPZ = aerobus::zpz<211>; using type =
04879
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<209»; }; // NOLINT
04880
                                           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»; }; // NOLINI
                                       template<> struct ConwayPolynomial<211, 7> { using ZPZ = aerobus::zpz<211>; using type =
04881
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<209»; }; // NOLINT template<> struct ConwayPolynomial<211, 8> { using ZPZ = aerobus::zpz<211>; using type =
04882
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<20>, ZPZV<27>, ZPZV<29>, ZPZV<29>, ZPZV<2»; }; //
                                          template<> struct ConwayPolynomial<211, 9> { using ZPZ = aerobus::zpz<211>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<139>, ZPZV<26>, ZPZV<209»;
                         }; // NOLINT
                         template<> struct ConwayPolynomial<211, 10> { using ZPZ = aerobus::zpz<211>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<30>, ZPZV<61>, ZPZV<4148>, ZPZV<87>, ZPZV<125>,
04884
                         ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<211, 11> { using ZPZ = aerobus::zpz<211>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<7>, ZPZV<209»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type :
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<50>, ZPZV<145>, ZPZV<126>, ZPZV<184>, ZPZV<84>, ZPZV<27>, ZPZV<284>, 
04887
                                           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>
04888
                                          template<> struct ConwayPolynomial<211, 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<0>, ZPZV<12>, ZPZV<20, ZPZV<2
                         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<17>, ZPZV<209»; }; //</pre>
                         NOLINT
04890
                                         template<> struct ConwayPolynomial<223, 1> { using ZPZ = aerobus::zpz<223>; using type =
                         POLYV<ZPZV<1>, ZPZV<220»; }; // NOLINT
                                           template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
                         POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; }; // NOLINT
04892
                                        template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, 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 =
04893
04894
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; }; // NOLINT
                                          template<> struct ConwayPolynomial<223, 6> { using ZPZ = aerobus::zpz<223>; using type =
04895
                         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 =
04896
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
04897
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<138>; };
04898
                                         template<> struct ConwayPolynomial<223, 9> { using ZPZ = aerobus::zpz<223>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<164>, ZPZV<64>, ZPZV<220»;
                         }; // NOLINT
04899
                                           template<> struct ConwayPolynomial<223, 10> { using ZPZ = aerobus::zpz<223>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<118>, ZPZV<177>, ZPZV<87>, ZPZV<99>, ZPZV<62>,
                         ZPZV<3»; }; // NOLINT</pre>
04900
                                        template<> struct ConwayPolynomial<223, 11> { using ZPZ = aerobus::zpz<223>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04901
                                        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<66>,
                         ZPZV<151>, ZPZV<213>, ZPZV<3»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<223, 13> { using ZPZ = aerobus::zpz<223>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<220»; };</pre>
                                                                                                                                                                                                        // NOLINT
                                          template<> struct ConwayPolynomial<223, 17> { using ZPZ = aerobus::zpz<223>; using type =
04903
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                // NOLINT
                                       template<> struct ConwayPolynomial<223, 19> { using ZPZ = aerobus::zpz<223>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         NOLINT
04905
                                          template<> struct ConwayPolynomial<227, 1> { using ZPZ = aerobus::zpz<227>; using type =
                         POLYV<ZPZV<1>, ZPZV<225»; }; // NOLINT
04906
                                       template<> struct ConwayPolynomial<227, 2> { using ZPZ = aerobus::zpz<227>; using type =
                        POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
04907
                        POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<227, 5> { using ZPZ = aerobus::zpz<227>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<25»; }; // NOLINT
04910
                                        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
04911
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<225»; }; // NOLINT
04912
                                        template<> struct ConwayPolynomial<227, 8> { using ZPZ = aerobus::zpz<227>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<151>, ZPZV<176>, ZPZV<106>, ZPZV<2»; }; //
                         NOLINT
04913
                                       template<> struct ConwayPolynomial<227, 9> { using ZPZ = aerobus::zpz<227>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<24>, ZPZV<24>, ZPZV<183>, ZPZV<225»;
                        }; // NOLINT template<> struct ConwayPolynomial<227, 10> { using ZPZ = aerobus::zpz<227>; using type
04914
                         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>
04915
                                          template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<225»; }; // NOLINT
                                          template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<99>, ZPZV<96>, ZPZV<96>, ZPZV<127>, ZPZV<142>, ZPZV<94>, ZPZV<22*; }; // NOLINT template<> struct ConwayPolynomial<227, 13> { using ZPZ = aerobus::zpz<227>; using type =
04917
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<225»; };</pre>
                                                                                                                                                                                                       // NOLINT
                                          template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>; ZPZV<0>; ZPZV<0>; ZPZV<0>; ZPZV<0>; ZPZV<0>, ZPZV<0 , ZPZ
                         NOLINT
                                          template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
                        POLYY<ZPZY<1>, ZPZY<223»; }; // NOLINT template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
04921
                        POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»; }; // NOLINT
                                          template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
04922
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; }; // NOLINT
                                       template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
04923
                        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<7>, ZPZV<162>, ZPZY<6»; }; // NOLINT template<> struct ConwayPolynomial<229, 5> { using ZPZ = aerobus::zpz<229>; using type =
04924
                        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 =
04925
                        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»; };
04927
                                       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»; }; //
                        NOLINT
                                        template<> struct ConwayPolynomial<229, 9> { using ZPZ = aerobus::zpz<229>; using type =
04928
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<117>, ZPZV<50>, ZPZV<223»;
                         }; // NOLINT
04929
                                          template<> struct ConwayPolynomial<229, 10> { using ZPZ = aerobus::zpz<229>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<185>, ZPZV<135>, ZPZV<158>, ZPZV<167>,
                        ZPZV<98>, ZPZV<6»; }; // NOLINT
  template<> struct ConwayPolynomial<229, 11> { using ZPZ = aerobus::zpz<229>; using type =
04930
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                             // NOLINT
                         ZPZV<2>, ZPZV<223»; };</pre>
04931
                                        template<> struct ConwayPolynomial<229, 12> { using ZPZ = aerobus::zpz<229>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<131>, ZPZV<140>, ZPZV<25>, ZPZV<26>, ZPZV<172>,
                         ZPZV<9>, ZPZV<145>, ZPZV<6»: }: // NOLINT
                                          template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type :
04932
                          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<47>, ZPZV<223»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type :
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
04934
```

```
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<228>, ZPZV<15>, ZPZV<223»; }; //</pre>
                            NOLINT
04935
                                             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 =
04936
                            POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
                                               template<> struct ConwayPolynomial<233, 3> { using ZPZ = aerobus::zpz<233>; using type =
 04937
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
 04938
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<233, 5> { using ZPZ = aerobus::zpz<233>; using type =
 04939
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<230»; }; // NOLINT
                                                template<> struct ConwayPolynomial<233, 6> { using ZPZ = aerobus::zpz<233>; using type =
 04940
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<122>, ZPZV<215>, ZPZV<32>, ZPZV<3»; }; // NOLINT
 04941
                                               template<> struct ConwayPolynomial<233, 7> { using ZPZ = aerobus::zpz<233>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230»; }; // NOLINT
                                             template<> struct ConwayPolynomial<233, 8> { using ZPZ = aerobus::zpz<233>; using type =
04942
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<202>, ZPZV<135>, ZPZV<181>, ZPZV<3»; }; //
                            template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<146>, ZPZV<230»;
                            }; // NOLINT
                                               template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type =
04944
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<28>, ZPZV<71>, ZPZV<102>, ZPZV<3>, ZPZV<48>,
                            ZPZV<3»; }; // NOLINT
                                           template<> struct ConwayPolynomial<233, 11> { using ZPZ = aerobus::zpz<233>; using type =
04945
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<5>, ZPZV<230»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<233, 12> { using ZPZ = aerobus::zpz<233>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<114>, ZPZV<114>, ZPZV<114>, ZPZV<115, ZPZV<
04946
                            ZPZV<216>, ZPZV<20>, ZPZV<3»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<233, 13> { using ZPZ = aerobus::zpz<233>; using type =
04947
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<14>, ZPZV<230»; }; // NOLINT
  template<> struct ConwayPolynomial<233, 17> { using ZPZ = aerobus::zpz<233>; using type =
04948
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230»; };</pre>
                                               template<> struct ConwayPolynomial<233, 19> { using ZPZ = aerobus::zpz<233>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<25>, ZPZV<230»; }; //</pre>
                            NOLINT
04950
                                              template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
                            POLYV<ZPZV<1>, ZPZV<232»; }; // NOLINT
                                               template<> struct ConwayPolynomial<239, 2> { using ZPZ = aerobus::zpz<239>; using type =
                            POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7»; }; // NOLINT
 04952
                                             template<> struct ConwayPolynomial<239, 3> { using ZPZ = aerobus::zpz<239>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<232»; }; // NOLINT template<> struct ConwayPolynomial<239, 4> { using ZPZ = aerobus::zpz<239>; using type =
04953
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7»; }; // NOLINT
                                               template<> struct ConwayPolynomial<239, 5> { using ZPZ = aerobus::zpz<239>; using type =
04954
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; }; // NOLINT
 04955
                                               template<> struct ConwayPolynomial<239, 6> { using ZPZ = aerobus::zpz<239>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<23>, ZPZV<60>, ZPZV<200>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<239, 7> { using ZPZ = aerobus::zpz<239>; using type =
04956
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<232»; }; // NOLINT 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»; };
04958
                                               template<> struct ConwayPolynomial<239, 9> { using ZPZ = aerobus::zpz<239>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<88>, ZPZV<232»; };
                            // NOLINT
04959
                                               template<> struct ConwayPolynomial<239, 10> { using ZPZ = aerobus::zpz<239>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<68>, ZPZV<226>, ZPZV<127>, ZPZV<108>, ZPZV<7»; }; // NOLINT
04960
                                            template<> struct ConwayPolynomial<239, 11> { using ZPZ = aerobus::zpz<239>; using type =
                             \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                            ZPZV<8>, ZPZV<232»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<239, 12> { using ZPZ = aerobus::zpz<239>; using type =
04961
                            POLYV<ZPZV<1), ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1, ZPZV<14>, ZPZV<14>, ZPZV<113>, ZPZV<182>, ZPZV<101>, ZPZV<216>, ZPZV<7»; }; // NOLINT
04962
                                            template<> struct ConwayPolynomial<239, 13> { using ZPZ = aerobus::zpz<239>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                               template<> struct ConwayPolynomial<239, 17> { using ZPZ = aerobus::zpz<239>; using type =
04963
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<232»; };</pre>
04964
                                              template<> struct ConwayPolynomial<239, 19> { using ZPZ = aerobus::zpz<239>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<232»; }; //</pre>
                            NOLINT
04965
                                               template<> struct ConwayPolynomial<241, 1> { using ZPZ = aerobus::zpz<241>; using type =
                            POLYV<ZPZV<1>, ZPZV<234»; }; // NOLINT
 04966
                                            template<> struct ConwayPolynomial<241, 2> { using ZPZ = aerobus::zpz<241>; using type =
                            POLYV<ZPZV<1>, ZPZV<238>, ZPZV<7»; }; // NOLINT
                                               \texttt{template<> struct ConwayPolynomial<241, 3> \{ using ZPZ = aerobus::zpz<241>; using type = 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241 + 241
 04967
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<234»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
04968
                       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<14>, ZPZV<152>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<241, 5> { using ZPZ = aerobus::zpz<241>; using type =
04969
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<234»; }; // NOLINT
                      template<> struct ConwayPolynomial<241, 6> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<6>, ZPZV<5>, ZPZV<7»; }; // NOLINT</pre>
 04970
                                       template<> struct ConwayPolynomial<241, 7> { using ZPZ = aerobus::zpz<241>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<234»; };
 04972
                                     template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<113>, ZPZV<212>, ZPZV<153>, ZPZV<7»; }; //
                       NOLINT
                                      template<> struct ConwayPolynomial<241, 9> { using ZPZ = aerobus::zpz<241>; using type =
04973
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<236>, ZPZV<125>, ZPZV<234»;
                        }; // NOLINT
 04974
                                     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
04975
                                      template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<3>, ZPZV<234»; };</pre>
                                                                                                                     // NOLINT
                       template<> struct ConwayPolynomial<241, 12> { using ZPZ = aerobus::zpz<241>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<10>, ZPZV<109>, ZPZV<168>, ZPZV<22>,
                        ZPZV<197>, ZPZV<17>, ZPZV<7»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type =
04977
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<241,
                                                                                                                                                                                                17> { using ZPZ = aerobus::zpz<241>; using type =
04978
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<234»; }; // NOLINT
template<> struct ConwayPolynomial<241, 19> { using ZPZ = aerobus::zpz<241>; using type =
04979
                        POLYY<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<1>, ZPZV<234w; }; //</pre>
                                      template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
04980
                       POLYV<ZPZV<1>, ZPZV<245»; }; // NOLINT
                                        template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
04981
                       POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; }; // NOLINT
                                       template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type =
 04983
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<200>, ZPZV<6»; }; // NOLINT
                       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
04984
04985
                                      template<> struct ConwayPolynomial<251, 6> { using ZPZ = aerobus::zpz<251>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<247>, ZPZV<151>, ZPZV<179>, ZPZV<6»; }; // NOLINT
 04986
                                       template<> struct ConwayPolynomial<251, 7> { using ZPZ = aerobus::zpz<251>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<45»; }; // NOLINT template<> struct ConwayPolynomial<251, 8> { using ZPZ = aerobus::zpz<251>; using type =
 04987
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<142>, ZPZV<215>, ZPZV<173>, ZPZV<6»; }; //
                        NOLINT
04988
                                       template<> struct ConwayPolynomial<251, 9> { using ZPZ = aerobus::zpz<251>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<106>, ZPZV<245»;
                        }; // NOLINT
04989
                                       template<> struct ConwayPolynomial<251, 10> { using ZPZ = aerobus::zpz<251>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPŽV<0>, ZPZV<0>, ZPZV<1>, ZPZV<138>, ZPZV<110>, ZPZV<45>, ZPŽV<34>,
                        ZPZV<149>, ZPZV<6»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<26>, ZPZV<245»; }; // NOLINT</pre>
04991
                                        template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type :
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<20>, ZPZV<20>, ZPZV<15>, ZPZV<20>, ZPZV<2
04992
                                       template<> struct ConwayPolynomial<251, 13> { using ZPZ = aerobus::zpz<251>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<245»; }; // NOLINT</pre>
04993
                                    template<> struct ConwayPolynomial<251, 17> { using ZPZ = aerobus::zpz<251>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>; ZPZV<0
04994
                        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+</pre>; }; //
04995
                                       template<> struct ConwayPolynomial<257, 1> { using ZPZ = aerobus::zpz<257>; using type =
                       POLYV<ZPZV<1>, ZPZV<254»; }; // NOLINT
                                       template<> struct ConwayPolynomial<257, 2> { using ZPZ = aerobus::zpz<257>; using type =
04996
                       POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3»; }; // NOLINT
                                      template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
 04997
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<254»; }; // NOLINT template<> struct ConwayPolynomial<257, 4> { using ZPZ = aerobus::zpz<257>; using type =
04998
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<187>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<257, 5> { using ZPZ = aerobus::zpz<257>; using type =
 04999
                       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 =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<62>, ZPZV<18>, ZPZV<138>, ZPZV<3»; }; // NOLINT
                                    template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type =
 05001
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25+; ; // NOLINT template<> struct ConwayPolynomial<257, 8> { using ZPZ = aerobus::2pz<257>; using type =
 05002
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<179>, ZPZV<140>, ZPZV<162>, ZPZV<3»; }; //
05003
                                     template<> struct ConwayPolynomial<257, 9> { using ZPZ = aerobus::zpz<257>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<201>, ZPZV<50>, ZPZV<254»;
                       }; // NOLINT
                                       template<> struct ConwayPolynomial<257, 10> { using ZPZ = aerobus::zpz<257>; using type =
05004
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<22>, ZPZV<22>, ZPZV<30>, ZPZV<30>
                       ZPZV<3»; }; // NOLINT
                                     template<> struct ConwayPolynomial<257, 11> { using ZPZ = aerobus::zpz<257>; using type =
05005
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<257, 12> { using ZPZ = aerobus::zpz<257>; using type =
05006
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<23>, ZPZV<225>, ZPZV<215>, ZPZV<2173>, ZPZV<249>, ZPZV<148>, ZPZV<20>, ZPZV<3»; }; // NOLINT
05007
                                    template<> struct ConwayPolynomial<257, 13> { using ZPZ = aerobus::zpz<257>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<257, 17> { using ZPZ = aerobus::zpz<257>; using type =
05008
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<254»; };</pre>
                                     template<> struct ConwayPolynomial<257, 19> { using ZPZ = aerobus::zpz<257>; using type
05009
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<10>, ZPZV<254»; }; //</pre>
                       NOLINT
05010
                                       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 =
05011
                       POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; }; // NOLINT
05012
                                       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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; }; // NOLINT
05013
                                       template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
05014
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<258»; }; // NOLINT
05015
                                       template<> struct ConwayPolynomial<263, 6> { using ZPZ = aerobus::zpz<263>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<250>, ZPZV<250>, ZPZV<255>, ZPZV<253>, j; // NOLINT template<> struct ConwayPolynomial<263, 7> { using ZPZ = aerobus::zpz<263>; using type
05016
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<258»; }; //
                                       template<> struct ConwayPolynomial<263, 8> { using ZPZ = aerobus::zpz<263>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<227>, ZPZV<170>, ZPZV<7>, ZPZV<5»; };
05018
                       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<61>, ZPZV<261>, ZPZV<29>, ZPZV<258»;</pre>
                      }; // 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>,
                                       template<> struct ConwayPolynomial<263, 11> { using ZPZ = aerobus::zpz<263>; using type =
05020
                       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 =
05021
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<172>, ZPZV<174>, ZPZV<162>, ZPZV<252>,
                        ZPZV<47>, ZPZV<45>, ZPZV<180>, ZPZV<5»; }; // NOLINT</pre>
05022
                                       template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
                       POLYV<ZPZV<1>, ZPZV<267»; }; // NOLINT
                                       template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
05023
                       POLYV<ZPZV<1>, ZPZV<268>, ZPZV<2»; }; // NOLINT
                                     template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
05024
                       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<9>, ZPZY<267»; }; // NOLINT template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
05025
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<269, 5> { using ZPZ = aerobus::zpz<269>; using type =
05026
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<267»; }; // NOLINT
                                        template<> struct ConwayPolynomial<269, 6> { using ZPZ = aerobus::zpz<269>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<101>, ZPZV<206>, ZPZV<2»; }; // NOLINT
05028
                                     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<6>, ZPZV<66>, ZPZV<267»; }; // NOLINT template<> struct ConwayPolynomial<269, 8> { using ZPZ = aerobus::zpz<269>; using type =
05029
                       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 =
05030
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2+, ZPZV<214>, ZPZV<267>, ZPZV<267>;
                       }; // NOLINT
05031
                                       template<> struct ConwayPolynomial<269, 10> { using ZPZ = aerobus::zpz<269>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<264>, ZPZV<243>, ZPZV<186>, ZPZV<61>, ZPZV<10>, ZPZV<20>, ZP
                                     template<> struct ConwayPolynomial<269, 11> { using ZPZ = aerobus::zpz<269>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<20>, ZPZV<267»; }; // NOLINT
05033
                                     template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2», ZPZV<165>, ZPZV<165>, ZPZV<63>, ZPZV<215>, ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2»; }; // NOLINT
                                       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 =
05035
                     POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
05036
```

```
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
                      template<> struct ConwayPolynomial<271, 5> { using ZPZ = aerobus::zpz<271>; using type =
05038
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT
            template<> struct ConwayPolynomial<271, 6> { using ZPZ = aerobus::zpz<271>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<207>, ZPZV<207>, ZPZV<81>, ZPZV<6»; }; // NOLINT
05039
                      template<> struct ConwayPolynomial<271, 7> { using ZPZ = aerobus::zpz<271>; using type =
05040
             POLYV<ZPZV<1>, ZPZV<0>, 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 =
05041
             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1, ZPZV<199>, ZPZV<114>, ZPZV<69>, ZPZV<6»; };
             NOLINT
                      template<> struct ConwayPolynomial<271, 9> { using ZPZ = aerobus::zpz<271>; using type =
05042
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<266>, ZPZV<186>, ZPZV<265»;
             }; // NOLINT
05043
                      template<> struct ConwayPolynomial<271, 10> { using ZPZ = aerobus::zpz<271>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<1>, ZPZV<133>, ZPZV<10>, ZPZV<256>, ZPZV<74>,
             ZPZV<126>, ZPZV<6»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
             ZPZV<10>, ZPZV<265»; }; // NOLINT</pre>
05045
                       template<> struct ConwayPolynomial<271, 12> { using ZPZ = aerobus::zpz<271>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162>, ZPZV<210>, ZPZV<116>, ZPZV<116>, ZPZV<205>, ZPZV<237>, ZPZV<256>, ZPZV<130>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
05046
             POLYV<ZPZV<1>, ZPZV<272»; }; // NOLINT
05047
                     template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
             POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; }; // NOLINT
                      template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
05048
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; }; // NOLINT template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
05049
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<5»; };
                                                                                                                                             // NOLINT
                      template<> struct ConwayPolynomial<277, 5> { using ZPZ = aerobus::zpz<277>; using type =
05050
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<272»; }; // NOLINT
05051
                      template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<9>, ZPZV<118>, ZPZV<5»; }; // NOLINT
                     template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type =
05052
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<272»; }; // NOLINT
                      template<> struct ConwayPolynomial<277, 8> { using ZPZ = aerobus::zpz<277>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<159>, ZPZV<176>, ZPZV<5»; }; //
             template<> struct ConwayPolynomial<277, 9> { using ZPZ = aerobus::zpz<277>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177>, ZPZV<110>, ZPZV<272»;</pre>
05054
             }; // 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>,
             template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05056
             ZPZV<5>, ZPZV<272»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<277, 12> { using ZPZ = aerobus::zpz<277>; using type =
05057
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<18>, ZPZV<218>, ZPZV<240>, ZPZV<40>,
              ZPZV<180>, ZPZV<115>, ZPZV<202>, ZPZV<5»; }; // NOLINT</pre>
05058
                      template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
             POLYV<ZPZV<1>, ZPZV<278»; }; // NOLINT
                      template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
05059
             POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; }; // NOLINT
                     template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<278»; }; // NOLINT
template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
05061
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<281, 5> { using ZPZ = aerobus::zpz<281>; using type =
05062
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<278»; }; // NOLINT
                       template<> struct ConwayPolynomial<281, 6> { using ZPZ = aerobus::zpz<281>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<151>, ZPZV<13>, ZPZV<27>, ZPZV<3»; }; // NOLINT
05064
                     template<> struct ConwayPolynomial<281, 7> { using ZPZ = aerobus::zpz<281>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2>8; }; // NOLINT template<> struct ConwayPolynomial<281, 8> { using ZPZ = aerobus::zpz<281>; using type =
05065
             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 =
05066
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<70>, 
             }; // NOLINT
05067
                      template<> struct ConwayPolynomial<281, 10> { using ZPZ = aerobus::zpz<281>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<145>, ZPZV<13>, ZPZV<138>, ZPZV<191>, ZPZV<3»; }; // NOLINT
                     template<> struct ConwayPolynomial<281, 11> { using ZPZ = aerobus::zpz<281>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
             ZPZV<36>, ZPZV<278»; }; // NOLINT
             template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<103>, ZPZV<103>, ZPZV<116>,
05069
             ZPZV<58>, ZPZV<28>, ZPZV<191>, ZPZV<38; }; // NOLINT template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
            POLYV<ZPZV<1>, ZPZV<280»; }; // NOLINT
                     template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
05071
            POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3»; }; // NOLINT
                     template<> struct ConwayPolynomial<283, 3> { using ZPZ = aerobus::zpz<283>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
                      template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; }; // NOLINT
                     template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
05074
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
            template<> struct ConwayPolynomial<283, 6> { using ZPZ = aerobus::zpz<283>; using type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<68>, ZPZV<73>, ZPZV<3»; }; // NOLINT
05075
                      template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type =
05076
             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<280»; }; // NOLINT template<> struct ConwayPolynomial<283, 8> { using ZPZ = aerobus::zpz<283>; using type =
05077
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<179>, ZPZV<32>, ZPZV<33>; }; //
             NOLINT
                     template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type
05078
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
             }; // NOLINT
05079
                      template<> struct ConwayPolynomial<283, 10> { using ZPZ = aerobus::zpz<283>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<271>, ZPZV<185>, ZPZV<68>, ZPZV<100>,
             ZPZV<219>, ZPZV<3»; }; // NOLINT</pre>
                      template<> struct ConwayPolynomial<283, 11> { using ZPZ = aerobus::zpz<283>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
             ZPZV<4>, ZPZV<280»; }; // NOLINT</pre>
05081
                      template<> struct ConwayPolynomial<283, 12> { using ZPZ = aerobus::zpz<283>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<8>, ZPZV<96>, ZPZV<229>, ZPZV<49>, ZPZV<14>, ZPZV<56>, ZPZV<3»; }; // NOLINT
05082
                      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 =
05083
             POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; }; // NOLINT
05084
                      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 =
05085
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2»; };
                                                                                                                                           // NOLINT
                      template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
05086
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
05087
                      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<2x); // NOLINT template<> struct ConwayPolynomial<293, 7> { using ZPZ = aerobus::zpz<293>; using type =
05088
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<291»; }; //
                      template<> struct ConwayPolynomial<293, 8> { using ZPZ = aerobus::zpz<293>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<2»; }; //
05090
             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<20>, ZPZV<208>, ZPZV<190>, ZPZV<291»;</pre>
             }; // 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<28>, ZPZV<28>, ZPZV<46>, ZPZV<184>, ZPZV<24>,
             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 =
05093
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<159>, ZPZV<210>, ZPZV<125>, ZPZV<212>,
             ZPZV<167>, ZPZV<144>, ZPZV<157>, ZPZV<2»; }; // NOLINT</pre>
05094
                      template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
             POLYV<ZPZV<1>, ZPZV<302»; }; // NOLINT
                      template<> struct ConwayPolynomial<307, 2> { using ZPZ = aerobus::zpz<307>; using type =
05095
             POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5»; }; // NOLINT
                     template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
05096
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<302»; }; // NOLINT template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
05097
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<307, 5> { using ZPZ = aerobus::zpz<307>; using type =
05098
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<302»; }; // NOLINT
                      template<> struct ConwayPolynomial<307, 6> { using ZPZ = aerobus::zpz<307>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<213>, ZPZV<172>, ZPZV<61>, ZPZV<5»; }; // NOLINT
                     template<> struct ConwayPolynomial<307, 7> { using ZPZ = aerobus::zpz<307>; using type =
05100
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<302»; }; // NOLINT template<> struct ConwayPolynomial<307, 8> { using ZPZ = aerobus::zpz<307>; using type =
05101
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<283>, ZPZV<232>, ZPZV<131>, ZPZV<5»; }; //
             NOLINT
                      template<> struct ConwayPolynomial<307, 9> { using ZPZ = aerobus::zpz<307>; using type =
05102
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<165>, ZPZV<70>, ZPZV<302»;
             }; // NOLINT
05103
                      template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
             POLYV<ZPZV<1>, ZPZV<294»; }; // NOLINT
                      template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
             POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; }; // NOLINT
05105
                     template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; };
                                                                                                                       // NOLINT
                      template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
05106
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294»; }; // NOLINT
05108
                     template<> struct ConwayPolynomial<311, 6> { using ZPZ = aerobus::zpz<311>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<167>, ZPZV<152>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<311, 7> { using ZPZ = aerobus::zpz<311>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<294»; }; // NOLINT
05109
```

```
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<217»; }; //
       NOLINT
      template<> struct ConwayPolynomial<311, 9> { using ZPZ = aerobus::zpz<311>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<287>, ZPZV<287>, ZPZV<294»;</pre>
05111
       }; // NOLINT
05112
            template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
       POLYV<ZPZV<1>, ZPZV<303»; }; // NOLINT
05113
           template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
      POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
05114
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; }; // NOLINT
           template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
05115
                                                                          // NOLINT
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<239>, ZPZV<10»; };
05116
           template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<303»; }; // NOLINT template<> struct ConwayPolynomial<313, 6> { using ZPZ = aerobus::zpz<313>; using type =
05117
      POLYVCZPZVC1>, ZPZVCO>, ZPZVCO>, ZPZVC196>, ZPZVC13A>, ZPZVC253>, ZPZVC253>, ZPZVC10»; }; // NOLINT template<> struct ConwayPolynomial<313, 7> { using ZPZ = aerobus::zpz<313>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<303»; };
           template<> struct ConwayPolynomial<313, 8> { using ZPZ = aerobus::zpz<313>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<306>, ZPZV<99>, ZPZV<106>, ZPZV<10»; }; //
       NOLINT
      template<> struct ConwayPolynomial<313, 9> { using ZPZ = aerobus::zpz<313>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<267>, ZPZV<300>, ZPZV<303»;</pre>
05120
       }; // NOLINT
           template<> struct ConwayPolynomial<317, 1> { using ZPZ = aerobus::zpz<317>; using type =
05121
      POLYV<ZPZV<1>, ZPZV<315»; }; // NOLINT
05122
            template<> struct ConwayPolynomial<317, 2> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
05123
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<7>, ZPZY<315»; }; // NOLINT template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
05124
      05125
           template<> struct ConwayPolynomial<317, 5> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<315»; // NOLINT
      template<> struct ConwayPolynomial<317, 6> { using ZPZ = aerobus::zpz<317>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<195>, ZPZV<156>, ZPZV<4>, ZPZV<2»; }; // NOLINT
05126
05127
            template<> struct ConwayPolynomial<317, 7> { using ZPZ = aerobus::zpz<317>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<315»; // NOLINT
05128
           template<> struct ConwayPolynomial<317, 8> { using ZPZ = aerobus::zpz<317>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<207>, ZPZV<85>, ZPZV<31>, ZPZV<2»; };
       NOLINT
05129
           template<> struct ConwayPolynomial<317, 9> { using ZPZ = aerobus::zpz<317>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<284>, ZPZV<286>, ZPZV<315»;
       }; // NOLINT
05130
           template<> struct ConwayPolynomial<331, 1> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<328»; }; // NOLINT
           template<> struct ConwayPolynomial<331, 2> { using ZPZ = aerobus::zpz<331>; using type =
05131
       POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; }; // NOLINT
05132
            template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<328»; }; // NOLINT
05133
           template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<290>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<331, 5> { using ZPZ = aerobus::zpz<331>; using type =
05134
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<328»; }; // NOLINT template<> struct ConwayPolynomial<331, 6> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<205>, ZPZV<159>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<331, 7> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<328»; }; // NOLIN template<> struct ConwayPolynomial<331, 8> { using ZPZ = aerobus::zpz<331>; using type =
05137
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<249>, ZPZV<308>, ZPZV<78>, ZPZV<3»; };
      NOLINT
           template<> struct ConwayPolynomial<331, 9> { using ZPZ = aerobus::zpz<331>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<194>, ZPZV<210>, ZPZV<328»;
       }; // NOLINT
05139
            template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<327»; }; // NOLINT
           template<> struct ConwayPolynomial<337, 2> { using ZPZ = aerobus::zpz<337>; using type =
05140
      POLYV<ZPZV<1>, ZPZV<332>, ZPZV<10»; }; // NOLINT
           template<> struct ConwayPolynomial<337, 3> { using ZPZ = aerobus::zpz<337>; using type =
05141
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT
           template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
0.5142
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<337, 5> { using ZPZ = aerobus::zpz<337>; using type =
05143
      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 =
05144
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<216>, ZPZV<127>, ZPZV<109>, ZPZV<109; }; // NOLINT template<> struct ConwayPolynomial<337, 7> { using ZPZ = aerobus::zpz<337>; using type
05145
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<327»; }; // NOLINT template<> struct ConwayPolynomial<337, 8> { using ZPZ = aerobus::zpz<337>; using type =
05146
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<331>, ZPZV<246>, ZPZV<251>, ZPZV<10»; }; //
05147
           template<> struct ConwayPolynomial<337, 9> { using ZPZ = aerobus::zpz<337>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<148>, ZPZV<98>, ZPZV<327»;
       }; // NOLINT
05148
           template<> struct ConwayPolynomial<347, 1> { using ZPZ = aerobus::zpz<347>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<345»; };
           template<> struct ConwayPolynomial<347, 2> { using ZPZ = aerobus::zpz<347>; using type =
      POLYY<ZPZV<1>, ZPZV<343>, ZPZV<2»; ); // NOLINT template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
05150
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
05151
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
05152
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<345»; }; // NOLINT
05153
           template<> struct ConwayPolynomial<347, 6> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<34>, ZPZV<26>, ZPZV<56>, ZPZV<56>, ZPZV<29; }; // NOLINT template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
05154
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<345»; };
           template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<187>, ZPZV<213>, ZPZV<117>, ZPZV<2»; }; //
           template<> struct ConwayPolynomial<347, 9> { using ZPZ = aerobus::zpz<347>; using type =
05156
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<235>, ZPZV<252>, ZPZV<345»;
       }; // NOLINT
           template<> struct ConwayPolynomial<349, 1> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<347»; }; // NOLINT
           template<> struct ConwayPolynomial<349, 2> { using ZPZ = aerobus::zpz<349>; using type =
05158
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
05159
      POLYV<ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<44, ZPZV<347»; }; // NOLINT template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2»; }; // NOLINT
05161
           template<> struct ConwayPolynomial<349, 5> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<347»; }; // NOLINT
           template<> struct ConwayPolynomial<349, 6> { using ZPZ = aerobus::zpz<349>; using type =
05162
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<135>, ZPZV<316>, ZPZV<316>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<349, 7> { using ZPZ = aerobus::zpz<349>; using type
05163
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347»; };
          template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type =
05164
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<328>, ZPZV<268>, ZPZV<2»; }; //
       NOLINT
      template<> struct ConwayPolynomial<349, 9> { using ZPZ = aerobus::zpz<349>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<290>, ZPZV<130>, ZPZV<347»;
05165
       }; // NOLINT
           template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type =
05166
      POLYV<ZPZV<1>, ZPZV<350»; }; // NOLINT
           template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
05167
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3»: }: // NOLINT
           template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
05168
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type =
05169
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
0.5170
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<350»; }; // NOLINT
           template<> struct ConwayPolynomial<353, 6> { using ZPZ = aerobus::zpz<353>; using type =
05171
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<215>, ZPZV<226>, ZPZV<295>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<353, 7> { using ZPZ = aerobus::zpz<353>; using type =
05172
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<350»; };
05173
           template<> struct ConwayPolynomial<353, 8> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<182>, ZPZV<26>, ZPZV<37>, ZPZV<3»; };
       NOLINT
           template<> struct ConwayPolynomial<353, 9> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<319>, ZPZV<49>, ZPZV<350»;
       }; // NOLINT
05175
           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 =
05176
      POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<352»; }; // NOLINT template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
05178
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<229>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<359, 5> { using ZPZ = aerobus::zpz<359>; using type =
05179
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; }; // NOLINT
05180
           template<> struct ConwayPolynomial<359, 6> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<309>, ZPZV<327>, ZPZV<327>, ZPZV<7»; }; // NOLINT
05181
          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<0>, ZPZV<1>, ZPZV<1>, ZPZV<352»; }; // NOLINT template<> struct ConwayPolynomial<359, 8> { using ZPZ = aerobus::zpz<359>; using type =
05182
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30+, ZPZV<301>, ZPZV<143>, ZPZV<271>, ZPZV<27»; }; //
           template<> struct ConwayPolynomial<359, 9> { using ZPZ = aerobus::zpz<359>; using type
05183
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<356>, ZPZV<165>, ZPZV<352»;
       }; // NOLINT
0.5184
           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 =
      POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6»; }; // NOLINT
05186
           template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<361»; }; // NOLINT template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6»; }; // NOLINT
0.5187
```

```
05188
           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
05189
           template<> struct ConwayPolynomial<367, 6> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<321>, ZPZV<324>, ZPZV<66; }; // NOLINT template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type
0.5190
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361»; };
                                                                                                         // NOLINT
           template<> struct ConwayPolynomial<367, 8> { using ZPZ = aerobus::zpz<367>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<335>, ZPZV<282>, ZPZV<50>, ZPZV<6»; };
      template<> struct ConwayPolynomial<367, 9> { using ZPZ = aerobus::zpz<367>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<213>, ZPZV<268>, ZPZV<361»;
05192
       }; // NOLINT
05193
           template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
       POLYV<ZPZV<1>, ZPZV<371»; }; // NOLINT
05194
           template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
05195
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<371»; };
                                                              // NOLINT
           template<> struct ConwayPolynomial<373, 4> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<373, 5> { using ZPZ = aerobus::zpz<373>; using type =
05197
      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 =
0.5198
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<83>, ZPZV<108>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type
05199
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<371»; }; //
05200
           template<> struct ConwayPolynomial<373, 8> { using ZPZ = aerobus::zpz<373>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<203>, ZPZV<219>, ZPZV<66>, ZPZV<2»; };
       NOLINT
05201
           template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<37), ZPZV<370>, ZPZV<371»;
       }; // NOLINT
           template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
05202
       POLYV<ZPZV<1>, ZPZV<377»; }; // NOLINT
05203
           template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
       POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial379, 3> { using ZPZ = aerobus::zpz<379>; using type =
05204
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377»; }; // NOLINT
05205
           template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2×; }; // NOLINT template<> struct ConwayPolynomial<379, 5> { using ZPZ = aerobus::zpz<379>; using type =
05206
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<377»; }; // NOLINT template<> struct ConwayPolynomial<379, 6> { using ZPZ = aerobus::zpz<379>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<374>, ZPZV<246>, ZPZV<246>, ZPZV<29*; }; // NOLINT
05207
           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»; };
05209
           template<> struct ConwayPolynomial<379, 8> { using ZPZ = aerobus::zpz<379>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<210>, ZPZV<194>, ZPZV<173>, ZPZV<2»; }; //
       NOLINT
05210
           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<36>, ZPZV<364, ZPZV<377»;
       }; // NOLINT
05211
           template<> struct ConwayPolynomial<383, 1> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<378»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 2> { using ZPZ = aerobus::zpz<383>; using type =
05212
      POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 5> { using ZPZ = aerobus::zpz<383>; using type =
05215
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
05216
           template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<8>, ZPZV<158>, ZPZV<5»; }; // NOLINT
05217
           template<> struct ConwayPolynomial<383, 7> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<378»; }; // NOLINT template<> struct ConwayPolynomial<383, 8> { using ZPZ = aerobus::zpz<383>; using type =
05218
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<281>, ZPZV<332>, ZPZV<296>, ZPZV<5»; }; //
       NOLINT
05219
           template<> struct ConwayPolynomial<383, 9> { using ZPZ = aerobus::zpz<383>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137>, ZPZV<76>, ZPZV<378»;
       }; // NOLINT
05220
           template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<387»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 2> { using ZPZ = aerobus::zpz<389>; using type =
05221
       POLYV<ZPZV<1>, ZPZV<379>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
05222
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387»; }; // NOLINT template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
05223
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<26>, ZPZV<266>, ZPZV<2%; }; // NOLINT template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
05224
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<387»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 6> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<339>, ZPZV<255>, ZPZV<2*; }; // NOLINT
05226
          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<2, ZPZV<2, ZPZV<2, ZPZV<387»; ); // NOLINT template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::2pz<389>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<351>, ZPZV<19>, ZPZV<290>, ZPZV<2»; }; //
05228
             template<> struct ConwayPolynomial<389, 9> { using ZPZ = aerobus::zpz<389>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<308>, ZPZV<387»;
        }; // NOLINT
05229
              template<> struct ConwayPolynomial<397, 1> { using ZPZ = aerobus::zpz<397>; using type =
        POLYV<ZPZV<1>, ZPZV<392»; }; // NOLINT
              template<> struct ConwayPolynomial<397, 2> { using ZPZ = aerobus::zpz<397>; using type =
05230
        POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5»; }; // NOLINT
05231
              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 =
05232
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392»; }; // NOLINT
              template<> struct ConwayPolynomial<397, 6> { using ZPZ = aerobus::zpz<397>; using type =
05234
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<274>, ZPZV<287>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 7> { using ZPZ = aerobus::zpz<397>; using type
05235
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<392»; }; // NOLINT
              template<> struct ConwayPolynomial<397, 8> { using ZPZ = aerobus::zpz<397>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<375>, ZPZV<255>, ZPZV<203>, ZPZV<5»; };
        NOLINT
05237
              template<> struct ConwayPolynomial<397, 9> { using ZPZ = aerobus::zpz<397>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<166>, ZPZV<252>, ZPZV<392»;
        }; // NOLINT
               template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
        POLYV<ZPZV<1>, ZPZV<398»; }; // NOLINT
              template<> struct ConwayPolynomial<401, 2> { using ZPZ = aerobus::zpz<401>; using type =
05239
        POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
05240
        POLYVCZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<398»; }; // NOLINT template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type =
05241
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<3»; }; // NOLINT
             template<> struct ConwayPolynomial<401, 5> { using ZPZ = aerobus::zpz<401>; using type =
05242
        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 =
05243
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<115>, ZPZV<51>, ZPZV<3»; ); // NOLINT 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<398»; };
              template<> struct ConwayPolynomial<401, 8> { using ZPZ = aerobus::zpz<401>; using type =
05245
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<380>, ZPZV<113>, ZPZV<164>, ZPZV<3»; }; //
        NOLINT
              template<> struct ConwayPolynomial<401, 9> { using ZPZ = aerobus::zpz<401>; using type =
05246
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<158>, ZPZV<398»;
05247
              template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
        POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
05248
        POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
05249
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
        05251
              template<> struct ConwayPolynomial<409, 5> { using ZPZ = aerobus::zpz<409>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<388»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 6> { using ZPZ = aerobus::zpz<409>; using type =
05252
        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
05253
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<4>, ZPZV<7>, ZPZV<40>, ZPZV<7>, zPZV<
05254
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<256>, ZPZV<69>, ZPZV<396>, ZPZV<396>, ZPZV<21»; }; //
        NOLINT
05255
              template<> struct ConwayPolynomial<409, 9> { using ZPZ = aerobus::zpz<409>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<81>, ZPZV<318>, ZPZV<211>, ZPZV<388»;
        }; // NOLINT
05256
              template<> struct ConwayPolynomial<419, 1> { using ZPZ = aerobus::zpz<419>; using type =
        POLYV<ZPZV<1>, ZPZV<417»; }; // NOLINT
              template<> struct ConwayPolynomial<419, 2> { using ZPZ = aerobus::zpz<419>; using type =
05257
        POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial4419, 3> { using ZPZ = aerobus::zpz<419>; using type =
05258
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type =
05259
        template<> struct ConwayPolynomial<419, 5> { using ZPZ = aerobus::zpz<419>; using type =
05260
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 6> { using ZPZ = aerobus::zpz<419>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<411>, ZPZV<33>, ZPZV<257>, ZPZV<2»; }; // NOLINT
05262
              template<> struct ConwayPolynomial<419, 7> { using ZPZ = aerobus::zpz<419>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4+, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 8> { using ZPZ = aerobus::zpz<419>; using type =
05263
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<234>, ZPZV<388>, ZPZV<151>, ZPZV<2»; }; //
        template<> struct ConwayPolynomial<419, 9> { using ZPZ = aerobus::zpz<419>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<386>, ZPZV<417»;</pre>
        }; // NOLINT
05265
              template<> struct ConwayPolynomial<421, 1> { using ZPZ = aerobus::zpz<421>; using type =
        POLYV<ZPZV<1>, ZPZV<419»; }; // NOLINT
```

```
05266
           template<> struct ConwayPolynomial<421, 2> { using ZPZ = aerobus::zpz<421>; using type =
       POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT
05267
           template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419»; }; // NOLINT
template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<421, 5> { using ZPZ = aerobus::zpz<421>; using type =
05268
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<419»; }; // NOLINT
05270
           template<> struct ConwayPolynomial<421, 6> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<111>, ZPZV<342>, ZPZV<41>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type =
05271
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<419»; }; // NOLINT
           template<> struct ConwayPolynomial<421, 8> { using ZPZ = aerobus::zpz<421>; using type
05272
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<389>, ZPZV<32>, ZPZV<77>, ZPZV<2»; };
       NOLINT
05273
           template<> struct ConwayPolynomial<421, 9> { using ZPZ = aerobus::zpz<421>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<394>, ZPZV<145>, ZPZV<419»;
       }; // NOLINT
            template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
       POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
            template<> struct ConwayPolynomial<431, 2> { using ZPZ = aerobus::zpz<431>; using type =
       POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7»; }; // NOLINT
           \texttt{template<>} \texttt{struct ConwayPolynomial<431, 3> \{ \texttt{using ZPZ = aerobus::zpz<431>; using type = 1.5 \}}
05276
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424»; }; // NOLINT template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
05277
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<323>, ZPZV<7»; }; // NOLINT
05278
           template<> struct ConwayPolynomial<431, 5> { using ZPZ = aerobus::zpz<431>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<424»; }; // NOLINT
05279
           template<> struct ConwayPolynomial<431, 6> { using ZPZ = aerobus::zpz<431>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<161>, ZPZV<202>, ZPZV<182>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<431, 7> { using ZPZ = aerobus::zpz<431>; using type
05280
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42*; };
           template<> struct ConwayPolynomial<431, 8> { using ZPZ = aerobus::zpz<431>; using type
05281
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<243>, ZPZV<286>, ZPZV<115>, ZPZV<7»; }; //
       NOLINT
05282
           template<> struct ConwayPolynomial<431, 9> { using ZPZ = aerobus::zpz<431>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<71>, ZPZV<329>, ZPZV<424*;
05283
            template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
       POLYV<ZPZV<1>, ZPZV<428»; }; // NOLINT
05284
           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 =
05285
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428»; }; // NOLINT template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<402>, ZPZV<5»; }; // NOLINT
05287
           template<> struct ConwayPolynomial<433, 5> { using ZPZ = aerobus::zpz<433>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<428»; }; // NOLINT
05288
           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
05289
           template<> struct ConwayPolynomial<433,
                                                         7> { using ZPZ = aerobus::zpz<433>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<428»; }; // NoLII template<> struct ConwayPolynomial<433, 8> { using ZPZ = aerobus::zpz<433>; using type :
05290
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347>, ZPZV<32>, ZPZV<39>, ZPZV<5»; }; //
       NOLINT
       template<> struct ConwayPolynomial<433, 9> { using ZPZ = aerobus::zpz<433>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<27>, ZPZV<232>, ZPZV<45>, ZPZV<428»;</pre>
05291
       }; // NOLINT
           template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
05292
       POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
05293
       POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
05294
       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<3>, ZPZV<424%; }; // NOLINT template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
05295
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
05296
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 6> { using ZPZ = aerobus::zpz<439>; using type =
05297
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<324>, ZPZV<190>, ZPZV<15»; };
           template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type =
05298
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT
05299
           template<> struct ConwayPolynomial<439, 8> { using ZPZ = aerobus::zpz<439>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<359>, ZPZV<296>, ZPZV<266>, ZPZV<15»; }; //
       NOLINT
           template<> struct ConwayPolynomial<439, 9> { using ZPZ = aerobus::zpz<439>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<342>, ZPZV<354>, ZPZV<424»;
       }; // NOLINT
05301
           template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
       POLYV<ZPZV<1>, ZPZV<441»; }; // NOLINT
           template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
05302
       POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<441»; };
                                                              // NOLINT
          template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
05304
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<4>, ZPZV<383>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<443>; using type =
05305
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<441»; };
           template<> struct ConwayPolynomial<443, 6> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<298>, ZPZV<218>, ZPZV<41>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<443, 7> { using ZPZ = aerobus::zpz<443>; using type =
05307
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6441»; }; // NOLINT template<> struct ConwayPolynomial<443, 8> { using ZPZ = aerobus::zpz<443>; using type =
05308
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<217>, ZPZV<290>, ZPZV<2»; }; //
           template<> struct ConwayPolynomial<443, 9> { using ZPZ = aerobus::zpz<443>; using type =
05309
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<125>, ZPZV<109>, ZPZV<441»;
       }; // NOLINT
05310
           template<> struct ConwayPolynomial<449, 1> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<446»; }; // NOLINT
            template<> struct ConwayPolynomial<449, 2> { using ZPZ = aerobus::zpz<449>; using type =
05311
       POLYV<ZPZV<1>, ZPZV<444>, ZPZV<3»; }; // NOLINT
05312
           template<> struct ConwayPolynomial<449, 3> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0, ZPZV<1>, ZPZV<446»; }; // NOLINT template<> struct ConwayPolynomial<449, 4> { using ZPZ = aerobus::zpz<449>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3»; }; // NOLINT
05313
           template<> struct ConwayPolynomial<449, 5> { using ZPZ = aerobus::zpz<449>; using type =
05314
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<446»; }; // NOLINT
05315
           template<> struct ConwayPolynomial<449, 6> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<293>, ZPZV<69>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<449, 7> { using ZPZ = aerobus::zpz<449>; using type =
05316
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<28>, ZPZV<446»; };
           template<> struct ConwayPolynomial<449, 8> { using ZPZ = aerobus::zpz<449>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361>, ZPZV<348>, ZPZV<124>, ZPZV<33»; }; //
       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<0>, ZPZV<6>, ZPZV<26>, ZPZV<26>, ZPZV<28>, ZPZV<446»; };</pre>
05318
       // NOLINT
05319
           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 =
05320
      POLYV<ZPZV<1>, ZPZV<454>, ZPZV<413»; }; // NOLINT template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
05321
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444»; }; // NOLINT
           template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13»; };
                                                                          // NOLINT
           template<> struct ConwayPolynomial<457, 5> { using ZPZ = aerobus::zpz<457>; using type =
05323
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<444»; }; // NOLINT
           template<> struct ConwayPolynomial<457, 6> { using ZPZ = aerobus::zpz<457>; using type =
05324
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<205>, ZPZV<389>, ZPZV<266>, ZPZV<13»; }; // NOLINT
05325
           template<> struct ConwayPolynomial<457, 7> { using ZPZ = aerobus::zpz<457>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<444»; }; // NOLINT
05326
           template<> struct ConwayPolynomial<457, 8> { using ZPZ = aerobus::zpz<457>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<365>, ZPZV<296>, ZPZV<412>, ZPZV<13»; }; //
       NOLINT
05327
           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<0>, ZPZV<354>, ZPZV<354>, ZPZV<8444»;
      }; // NOLINT template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<459»; }; // NOLINT template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
05329
      POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial4461, 3> { using ZPZ = aerobus::zpz<461>; using type =
05330
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459»; }; // NOLINT
           template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
05331
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2»; }; // NOLINT
05332
           template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<459»; }; // NOLINT template<> struct ConwayPolynomial<461, 6> { using ZPZ = aerobus::zpz<461>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<439>, ZPZV<432>, ZPZV<329>, ZPZV<2»; }; // NOLINT
05333
           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<5>, ZPZV<459»; };
05335
          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»; }; //
       NOLINT
           template<> struct ConwayPolynomial<461, 9> { using ZPZ = aerobus::zpz<461>; using type =
05336
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<210>, ZPZV<216>, ZPZV<459»;
       }; // NOLINT
05337
           template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<460»; }; // NOLINT
           template<> struct ConwayPolynomial<463, 2> { using ZPZ = aerobus::zpz<463>; using type =
05338
      POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<460»; }; // NOLINT
05340
           template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
05341
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<460»; }; // NOLINT
           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 =
05343
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, 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<341>, ZPZV<396>, ZPZV<396>, ZPZV<39; //
05344
```

```
NOLINT
         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»;
         }; // NOLINT
05346
               template<> struct ConwayPolynomial<467, 1> { using ZPZ = aerobus::zpz<467>; using type =
         POLYV<ZPZV<1>, ZPZV<465»; }; // NOLINT
               template<> struct ConwayPolynomial<467, 2> { using ZPZ = aerobus::zpz<467>; using type =
         POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
05348
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
05349
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<467, 5> { using ZPZ = aerobus::zpz<467>; using type =
05350
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
05351
              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 =
05352
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<318>, ZPZV<413>, ZPZV<289>, ZPZV<2*; }; //
05354
               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
05355
               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 =
05356
         POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; }; // NOLINT
05357
               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 =
05358
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13»; };
                                                                                                  // NOLINT
               template<> struct ConwayPolynomial<479, 5> { using ZPZ = aerobus::zpz<479>; using type =
05359
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<466»; }; // NOLINT
05360
               template<> struct ConwayPolynomial<479, 6> { using ZPZ = aerobus::zpz<479>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<243>, ZPZV<287>, ZPZV<334>, ZPZV<13»; }; // NOLINT
               template<> struct ConwayPolynomial<479, 7> { using ZPZ = aerobus::zpz<479>; using type
05361
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44, ZPZV<466»; }; //
               template<> struct ConwayPolynomial<479, 8> { using ZPZ = aerobus::zpz<479>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<247>, ZPZV<440>, ZPZV<17>, ZPZV<13»; }; //
         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»; };
05363
         // NOLINT
               template<> struct ConwayPolynomial<487, 1> { using ZPZ = aerobus::zpz<487>; using type =
         POLYV<ZPZV<1>, ZPZV<484»; }; // NOLINT
05365
              template<> struct ConwayPolynomial<487, 2> { using ZPZ = aerobus::zpz<487>; using type =
         POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
05366
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; }; // NOLINT template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type =
05367
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3»; }; // NOLINT
05368
               template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
         POLYY<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 =
05369
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<450>, 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<0>, ZPZV<0>, ZPZV<7>, ZPZV<484»; };
              template<> struct ConwayPolynomial<487, 8> { using ZPZ = aerobus::zpz<487>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<249>, ZPZV<137>, ZPZV<3»; }; //
         NOLINT
         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»;
05372
         }; // NOLINT
              template<> struct ConwayPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
05373
         POLYV<ZPZV<1>, ZPZV<489»; }; // NOLINT
05374
               template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
         POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial4491, 3> { using ZPZ = aerobus::zpz<491>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type =
05376
         \label{eq:polyv} \mbox{PDLYV<2PZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360>, ZPZV<2*; }; // \mbox{NOLINT}
               template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
05377
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT
         template<> struct ConwayPolynomial<491, 6> { using ZPZ = aerobus::zpz<491>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<369>, ZPZV<402>, ZPZV<125>, ZPZV<2»; }; // NOLINT
05378
               template<> struct ConwayPolynomial<491, 7> { using ZPZ = aerobus::zpz<491>; using type
05379
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 8> { using ZPZ = aerobus::zpz<491>; using type =
05380
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378>, ZPZV<372>, ZPZV<216>, ZPZV<2»; }; //
         NOLINT
05381
               template<> struct ConwayPolynomial<491, 9> { using ZPZ = aerobus::zpz<491>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<453>, ZPZV<489»;
         }; // NOLINT
05382
               template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
        POLYV<ZPZV<1>, ZPZV<492»; }; // NOLINT
05383
              template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; };
               template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
05385
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
05386
         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 =
05387
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<407>, ZPZV<191>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<499, 7> { using ZPZ = aerobus::zpz<499>; using type =
05388
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 8> { using ZPZ = aerobus::zpz<499>; using type =
05389
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<309>, ZPZV<200>, ZPZV<20»; }; //
05390
              template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<491>, ZPZV<222>, ZPZV<492»;
         }; // NOLINT
05391
               template<> struct ConwayPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
         POLYV<ZPZV<1>, ZPZV<498»; }; // NOLINT
               template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
         POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; }; // NOLINT
05393
               template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<498»; }; // NOLINT
template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<325>, ZPZV<5»; }; // NOLINT
05394
               template<> struct ConwayPolynomial<503, 5> { using ZPZ = aerobus::zpz<503>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<498»; }; // NOLINT
05396
               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
05397
              template<> struct ConwayPolynomial<503, 7> { using ZPZ = aerobus::zpz<503>; using type =
        POLYVCZPZV<1>, ZPZV<0>, ZPZV<0
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<441>, ZPZV<203>, ZPZV<316>, ZPZV<5»; }; //
         template<> struct ConwayPolynomial<503, 9> { using ZPZ = aerobus::zpz<503>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<158>, ZPZV<337>, ZPZV<498»;</pre>
05399
         }; // NOLINT
               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 =
05401
         POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
05402
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
05403
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2»; }; // NOLINT
05404
               template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
05405
              template<> struct ConwayPolynomial<509, 6> { using ZPZ = aerobus::zpz<509>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<350>, ZPZV<232>, ZPZV<41>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<509, 7> { using ZPZ = aerobus::zpz<509>; using type =
05406
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<507»; };
               template<> struct ConwayPolynomial<509, 8> { using ZPZ = aerobus::zpz<509>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<420>, ZPZV<473>, ZPZV<382>, ZPZV<2»; }; //
              template<> struct ConwayPolynomial<509, 9> { using ZPZ = aerobus::zpz<509>; using type =
05408
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<507»;
         }; // NOLINT
    template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
05409
         POLYV<ZPZV<1>, ZPZV<518»; }; // NOLINT
               template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
05410
         POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
05411
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518»; }; // NOLINT template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<509>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<521, 5> { using ZPZ = aerobus::zpz<521>; using type =
05413
         template<> struct ConwayPolynomial<521, 6> { using ZPZ = aerobus::zpz<521>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<315>, ZPZV<153>, ZPZV<280>, ZPZV<3»; }; // NOLINT
05414
05415
               template<> struct ConwayPolynomial<521, 7> { using ZPZ = aerobus::zpz<521>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<518»; }; // NOLIN template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type =
05416
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<462>, ZPZV<407>, ZPZV<312>, ZPZV<3»; }; //
         NOLINT
              template<> struct ConwayPolynomial<521, 9> { using ZPZ = aerobus::zpz<521>; using type =
05417
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<181>, ZPZV<483>, ZPZV<518»;
05418
               template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
         POLYV<ZPZV<1>, ZPZV<521»; }; // NOLINT
05419
               template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
         POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
        POLYV<ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
05421
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
05422
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<521»; }; // NOLINT
```

```
05423
               template<> struct ConwayPolynomial<523, 6> { using ZPZ = aerobus::zpz<523>; using type =
         POLYY<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 =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 8> { using ZPZ = aerobus::zpz<523>; using type =
05425
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<518>, ZPZV<184>, ZPZV<380>, ZPZV<2»; }; //
               template<> struct ConwayPolynomial<523, 9> { using ZPZ = aerobus::zpz<523>; using type
05426
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<342>, ZPZV<345>, ZPZV<521»;
         }; // NOLINT
05427
               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 =
05428
         POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; }; // NOLINT
05429
              template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539»; ); // NOLINT
template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
05430
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<539»; }; // NOLINT
               template<> struct ConwayPolynomial<541, 6> { using ZPZ = aerobus::zpz<541>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<239>, ZPZV<320>, ZPZV<69>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<541, 7> { using ZPZ = aerobus::zpz<541>; using type =
05433
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<539»; }; // NOLINT template<> struct ConwayPolynomial<541, 8> { using ZPZ = aerobus::zpz<541>; using type =
05434
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<376>, ZPZV<108>, ZPZV<113>, ZPZV<2»; }; //
         template<> struct ConwayPolynomial<541, 9> { using ZPZ = aerobus::zpz<541>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<340>, ZPZV<318>, ZPZV<539»;
05435
         }; // NOLINT
               template<> struct ConwayPolynomial<547, 1> { using ZPZ = aerobus::zpz<547>; using type =
05436
         POLYV<ZPZV<1>, ZPZV<545»; }; // NOLINT
               template<> struct ConwayPolynomial<547, 2> { using ZPZ = aerobus::zpz<547>; using type =
05437
         POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<547, 3> { using ZPZ = aerobus::zpz<547>; using type =
05438
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 4> { using ZPZ = aerobus::zpz<547>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<334>, ZPZV<2»; }; // NOLINT
05439
05440
               template<> struct ConwayPolynomial<547, 5> { using ZPZ = aerobus::zpz<547>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<545»; }; // NOLINT
         template<> struct ConwayPolynomial<547, 6> { using ZPZ = aerobus::zpz<547>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<547, 7> { using ZPZ = aerobus::zpz<547>; using type =
05441
05442
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<545»; };
               template<> struct ConwayPolynomial<547, 8> { using ZPZ = aerobus::zpz<547>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2»; }; //
         template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<263>, ZPZV<545»;</pre>
05444
         }; // NOLINT
               template<> struct ConwayPolynomial<557, 1> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<555»; }; // NOLINT
05446
               template<> struct ConwayPolynomial<557, 2> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
05447
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<555»; }; // NOLINT template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<557, 5> { using ZPZ = aerobus::zpz<557>; using type =
05449
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<555»; }; // NOLINT
              template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type =
05450
         POLYYCZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<202>, ZPZV<253>, ZPZV<253>, ZPZV<253>; // NOLINT template<> struct ConwayPolynomial<557, 7> { using ZPZ = aerobus::zpz<557>; using type
05451
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<555»; }; //
05452
               template<> struct ConwayPolynomial<557, 8> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<384>, ZPZV<113>, ZPZV<2»; }; //
         NOLINT
05453
              template<> struct ConwayPolynomial<557, 9> { using ZPZ = aerobus::zpz<557>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<1>, ZPZV<456>, ZPZV<434>, ZPZV<555»;
         }; // NOLINT
               template<> struct ConwayPolynomial<563, 1> { using ZPZ = aerobus::zpz<563>; using type =
05454
         POLYV<ZPZV<1>, ZPZV<561»; }; // NOLINT
               template<> struct ConwayPolynomial<563, 2> { using ZPZ = aerobus::zpz<563>; using type =
05455
         POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
05456
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; }; // NOLINT
               template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
05457
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<20>, ZPZV<399>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<563, 5> { using ZPZ = aerobus::zpz<563>; using type =
05458
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<561»; }; // NOLINT
        template<> struct ConwayPolynomial</br>
template</br>
template</br>
template</br>
template
temp
05459
               template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz<563>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<561»; };
05461
              template<> struct ConwayPolynomial<563, 8> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<176>, ZPZV<509>, ZPZV<2>; }; //
         NOLTNT
```

```
template<> struct ConwayPolynomial<563, 9> { using ZPZ = aerobus::zpz<563>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<15>, ZPZV<19>, ZPZV<561»; };
       // NOLINT
05463
           template<> struct ConwayPolynomial<569, 1> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<566»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 2> { using ZPZ = aerobus::zpz<569>; using type =
05464
      POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
05465
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
05466
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<569, 5> { using ZPZ = aerobus::zpz<569>; using type =
05467
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT
            template<> struct ConwayPolynomial<569, 6> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<50>, ZPZV<263>, ZPZV<480>, ZPZV<3»; }; // NOLINT
05469
           template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<566»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 8> { using ZPZ = aerobus::zpz<569>; using type =
05470
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<527>, ZPZV<173>, ZPZV<241>, ZPZV<241>, ZPZV<3»; }; //
      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>;
       }; // NOLINT
           template<> struct ConwayPolynomial<571, 1> { using ZPZ = aerobus::zpz<571>; using type =
05472
      POLYV<ZPZV<1>, ZPZV<568»; }; // NOLINT
            template<> struct ConwayPolynomial<571, 2> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
05474
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<58»; }; // NOLINT
template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<571, 5> { using ZPZ = aerobus::zpz<571>; using type =
05475
05476
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<568»; }; // NOLINT
05477
           template<> struct ConwayPolynomial<571, 6> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<221>, ZPZV<295>, ZPZV<3>; }; // NOLINT template<> struct ConwayPolynomial<571, 7> { using ZPZ = aerobus::zpz<571>; using type =
05478
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 8> { using ZPZ = aerobus::zpz<571>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<363>, ZPZV<119>, ZPZV<371>, ZPZV<3»; };
05480
           template<> struct ConwayPolynomial<571, 9> { using ZPZ = aerobus::zpz<571>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<545>, ZPZV<179>, ZPZV<568»;
       }; // NOLINT
05481
            template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<572»; }; // NOLINT
05482
            template<> struct ConwayPolynomial<577, 2> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
05483
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572»; }; // NOLINT
           template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
05484
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<5»; };
                                                                          // NOLINT
            template<> struct ConwayPolynomial<577, 5> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<572»; }; // NOLINT
      template<> struct ConwayPolynomial<577, 6> { using ZPZ = aerobus::zpz<577>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<450>, ZPZV<25>, ZPZV<283>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<577, 7> { using ZPZ = aerobus::zpz<577>; using type =
05486
05487
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<572»; }; //
           template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type =
05488
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<3»; }; //
       NOLINT
05489
           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
05490
            template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<585»; }; // NOLINT
05491
           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 =
05492
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585»; }; // NOLINT
            template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
05493
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<444>, ZPZV<2»; }; // NOLINT
05494
           template<> struct ConwayPolynomial<587, 5> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<585»; }; // NOLINT template<> struct ConwayPolynomial<587, 6> { using ZPZ = aerobus::zpz<587>; using type =
05495
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<204-, ZPZV<204-, ZPZV<26>, ZPZV<226>, ZPZV<28-; }; // NOLINT template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<585»; };
05497
           template<> struct ConwayPolynomial<587, 8> { using ZPZ = aerobus::zpz<587>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<492>, ZPZV<444>, ZPZV<91>, ZPZV<2»; };
      NOLINT
           template<> struct ConwayPolynomial<587, 9> { using ZPZ = aerobus::zpz<587>; using type =
05498
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<333>, ZPZV<55>, ZPZV<585»;
       }; // NOLINT
05499
           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 =
05500
       POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3»; }; // NOLINT
```

```
05501
                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 =
05502
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<593, 5> { using ZPZ = aerobus::zpz<593>; using type =
05503
         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 =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<345>, ZPZV<65>, ZPZV<478>, ZPZV<3»; }; // NOLINI
               template<> struct ConwayPolynomial<593, 7> { using ZPZ = aerobus::zpz<593>; using type =
05505
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<590»; };
               template<> struct ConwayPolynomial<593, 8> { using ZPZ = aerobus::zpz<593>; using type =
05506
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<350>, ZPZV<291>, ZPZV<495>, ZPZV<495), ZPZV<3»; }; //
         NOLINT
               template<> struct ConwayPolynomial<593, 9> { using ZPZ = aerobus::zpz<593>; using type =
05507
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223>, ZPZV<523>, ZPZV<590»;
         }; // NOLINT
05508
               template<> struct ConwayPolynomial<599, 1> { using ZPZ = aerobus::zpz<599>; using type =
         POLYV<ZPZV<1>, ZPZV<592»; }; // NOLINT
               template<> struct ConwayPolynomial<599, 2> { using ZPZ = aerobus::zpz<599>; using type =
         POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
                template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
05510
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<592»; }; // NOLINT template<> struct ConwayPolynomial<599, 4> { using ZPZ = aerobus::zpz<599>; using type =
05511
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<599, 5> { using ZPZ = aerobus::zpz<599>; using type =
05512
         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 =
05513
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type
05514
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<592»; }; // NOLINT
               template<> struct ConwayPolynomial<599, 8> { using ZPZ = aerobus::zpz<599>; using type =
05515
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<440>, ZPZV<37>, ZPZV<124>, ZPZV<7»; };
              template<> struct ConwayPolynomial<599, 9> { using ZPZ = aerobus::zpz<599>; using type =
05516
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<9>, ZPZV<592»;
         }; // NOLINT
         template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type = POLYV<ZPZV<1>, ZPZV<594»; }; // NOLINT
05517
                template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
         POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
05519
               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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; }; // NOLINT
05520
               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
05522
               template<> struct ConwayPolynomial<601, 6> { using ZPZ = aerobus::zpz<601>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<440>, ZPZV<49>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<601, 7> { using ZPZ = aerobus::zpz<601>; using type =
05523
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>4, NOLINT
               template<> struct ConwayPolynomial<601, 8> { using ZPZ = aerobus::zpz<601>; using type
05524
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>0>, ZPZV<550>, ZPZV<241>, ZPZV<490>, ZPZV<7»; }; //
         NOLINT
         template<> struct ConwayPolynomial<601, 9> { using ZPZ = aerobus::zpz<601>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<487>, ZPZV<590>, ZPZV<594»;</pre>
05525
         }; // NOLINT
                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 =
05527
         POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
05528
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<604»; }; // NOLINT template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
05529
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<607, 5> { using ZPZ = aerobus::zpz<607>; using type =
05530
         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 =
05531
05532
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
               template<> struct ConwayPolynomial<607, 8> { using ZPZ = aerobus::zpz<607>; using type
05533
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<468>, ZPZV<35>, ZPZV<449>, ZPZV<3*; };
         NOLINT
               template<> struct ConwayPolynomial<607, 9> { using ZPZ = aerobus::zpz<607>; using type =
05534
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<444>, ZPZV<429>, ZPZV<604»;
         }; // NOLINT
05535
               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 =
05536
         POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
05537
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<611»; }; // NOLINT
               template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2»; }; // NOLINT
        template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<611»; }; // NOLINT</pre>
05539
               template<> struct ConwayPolynomial<613, 6> { using ZPZ = aerobus::zpz<613>; using type =
05540
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<60>, ZPZV<609>, ZPZV<595>, ZPZV<601>, ZPZV<2»; };
           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<0>, ZPZV<6>, ZPZV<611»; }; // NOLINT
          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»; }; //
      NOLTNT
           template<> struct ConwayPolynomial<613, 9> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53, ZPZV<513>, ZPZV<536>, ZPZV<611»;
      }; // NOLINT
05544
           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 =
05545
      POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<614»; }; // NOLINT
           template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type =
05547
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<617, 5> { using ZPZ = aerobus::zpz<617>; using type =
05548
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<614»; }; // NOLINT
           template<> struct ConwayPolynomial<617, 6> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<318>, ZPZV<595>, ZPZV<310>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<617, 7> { using ZPZ = aerobus::zpz<617>; using type =
05550
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<614»; }; // NOLINT template<> struct ConwayPolynomial<617, 8> { using ZPZ = aerobus::zpz<617>; using type =
05551
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<501>, ZPZV<501>, ZPZV<155>, ZPZV<3»; }; //
           template<> struct ConwayPolynomial<617, 9> { using ZPZ = aerobus::zpz<617>; using type =
05552
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<543>, ZPZV<614»;
      }; // NOLINT
05553
           template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<617»; }; // NOLINT
05554
           template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; }; // NOLINT
05555
          template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617»; }; // NOLINT template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
05556
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492>, ZPZV<2»; }; // NOLINT 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 =
05558
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<468>, ZPZV<347>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<619, 7> { using ZPZ = aerobus::zpz<619>; using type =
05559
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<617»; }; // NOLINT
           template<> struct ConwayPolynomial<619, 8> { using ZPZ = aerobus::zpz<619>; using type =
05560
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<416>, ZPZV<383>, ZPZV<225>, ZPZV<2»; }; //
      NOLINT
05561
           template<> struct ConwayPolynomial<619, 9> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<510>, ZPZV<617»;
      }; // NOLINT
05562
           template<> struct ConwavPolynomial<631, 1> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<628»; }; // NOLINT
           template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3»; }; // NOLINT
05564
           template<> struct ConwayPolynomial<631, 3> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<628; }; // NOLINT template<> struct ConwayPolynomial<631, 4> { using ZPZ = aerobus::zpz<631>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<3»; }; // NOLINT
05565
           template<> struct ConwayPolynomial<631, 5> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
05567
           template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<516>, ZPZV<541>, ZPZV<106>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<631, 7> { using ZPZ = aerobus::zpz<631>; using type
05568
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
           template<> struct ConwayPolynomial<631, 8> { using ZPZ = aerobus::zpz<631>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<3»; }; //
      NOLINT
05570
      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<296>, ZPZV<413>, ZPZV<413>, ZPZV<413>, ZPZV<413>, ZPZV<413</pre>
      }; // NOLINT
05571
           template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<638»; }; // NOLINT
05572
          template<> struct ConwayPolynomial<641, 2> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
05573
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<4>, ZPZV<638»; }; // NOLINT template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<3»; }; // NOLINT
05575
           template<> struct ConwayPolynomial<641, 5> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<638»; }; // NOLINT
05576
           template<> struct ConwayPolynomial<641, 6> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<557>, ZPZV<294>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<641,
                                                        7> { using ZPZ = aerobus::zpz<641>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<638s; }; // NOLI template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>; using type
05578
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<33; }; //
05579
           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»;
05580
                 template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
          POLYV<ZPZV<1>, ZPZV<632»; }; // NOLINT
                  template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
05581
          POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11»; }; // NOLINT
                  template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<632»; }; // NOLINT
                 template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
05583
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<600>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<643, 5> { using ZPZ = aerobus::zpz<643>; using type =
05584
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<632»; }; // NOLINT
                  template<> struct ConwayPolynomial<643, 6> { using ZPZ = aerobus::zpz<643>; using type =
05585
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<345>, ZPZV<412>, ZPZV<293>, ZPZV<11»; }; // NOLIN
05586
                 template<> struct ConwayPolynomial<643, 7> { using ZPZ = aerobus::zpz<643>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<632»; }; // NOLINT template<> struct ConwayPolynomial<643, 8> { using ZPZ = aerobus::zpz<643>; using type =
05587
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<631>, ZPZV<573>, ZPZV<569>, ZPZV<11»; }; //
                 template<> struct ConwayPolynomial<643, 9> { using ZPZ = aerobus::zpz<643>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZP
          }; // NOLINT
05589
                 template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
          POLYV<ZPZV<1>, ZPZV<642»; }; // NOLINT
05590
                  template<> struct ConwayPolynomial<647, 2> { using ZPZ = aerobus::zpz<647>; using type =
          POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5»; }; // NOLINT
                  template<> struct ConwayPolynomial<647, 3> { using ZPZ = aerobus::zpz<647>; using type =
05591
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<642»; }; // NOLINT template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
05592
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5»; }; // NOLINT

template<> struct ConwayPolynomial<647, 5> { using ZPZ = aerobus::zpz<647>; using type =
05593
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<642»; }; // NOLINT
                  template<> struct ConwayPolynomial<647, 6> { using ZPZ = aerobus::zpz<647>; using type =
05594
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<308>, ZPZV<385>, ZPZV<642>, ZPZV<5»; }; // NOLINI
05595
                  template<> struct ConwayPolynomial<647, 7> { using ZPZ = aerobus::zpz<647>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<642»; }; // NOLINT
                 template<> struct ConwayPolynomial<647, 8> { using ZPZ = aerobus::zpz<647>; using type =
05596
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<603>, ZPZV<259>, ZPZV<271>, ZPZV<27»; }; //
                 template<> struct ConwayPolynomial<647, 9> { using ZPZ = aerobus::zpz<647>; using type =
05597
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<561>, ZPZV<561>, ZPZV<642»;
          }; // NOLINT
                  template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
05598
          POLYV<ZPZV<1>, ZPZV<651»; }; // NOLINT
                  template<> struct ConwayPolynomial<653, 2> { using ZPZ = aerobus::zpz<653>; using type =
          POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2»; }; // NOLINT
05600
                 template<> struct ConwayPolynomial<653, 3> { using ZPZ = aerobus::zpz<653>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<651»; }; // NOLINT
template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
05601
05602
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651»; }; // NOLINT
05603
                  template<> struct ConwayPolynomial<653, 6> { using ZPZ = aerobus::zpz<653>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<45>, ZPZV<220>, ZPZV<242>, ZPZV<224>; ; // NOLINT template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type =
05604
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<385>, ZPZV<18>, ZPZV<296>, ZPZV<2»; };
05606
                  template<> struct ConwayPolynomial<653, 9> { using ZPZ = aerobus::zpz<653>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<651»;
          }; // NOLINT
05607
                  template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
          POLYV<ZPZV<1>, ZPZV<657»; }; // NOLINT
05608
                 template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
          POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2»; }; // NOLINT
05609
                  template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<657%; }; // NOLINT template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
05610
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
05611
          05612
                  template<> struct ConwayPolynomial<659, 6> { using ZPZ = aerobus::zpz<659>; using type =
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<371>, ZPZV<105>, ZPZV<223>, ZPZV<2»; }; // NOLINT
                                                                                         7> { using ZPZ = aerobus::zpz<659>; using type
05613
                  template<> struct ConwayPolynomial<659,
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<657»; }; //
                 template<> struct ConwayPolynomial<659, 8> { using ZPZ = aerobus::zpz<659>; using type
05614
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<358>, ZPZV<246>, ZPZV<90>, ZPZV<20; };
          NOLINT
05615
          template<> struct ConwayPolynomial<659, 9> { using ZPZ = aerobus::zpz<659>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592>, ZPZV<592>, ZPZV<657»;</pre>
          }; // NOLINT
    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 =
05617
          POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<659»; }; // NOLINT
        template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
05620
        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<2»; }; // NOLINT
05621
               template<> struct ConwayPolynomial<661, 7> { using ZPZ = aerobus::zpz<661>; using type
05622
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<659»; }; // NOLINT template<> struct ConwayPolynomial<661, 8> { using ZPZ = aerobus::zpz<661>; using type =
05623
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<61>, ZPZV<85>, ZPZV<72>, ZPZV<72>, ZPZV<2»; };
         NOLINT
              template<> struct ConwayPolynomial<661, 9> { using ZPZ = aerobus::zpz<661>; using type =
05624
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<389>, ZPZV<220>, ZPZV<659»;
         }; // NOLINT
05625
               template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
        POLYV<ZPZV<1>, ZPZV<668»; }; // NOLINT
               template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
05626
         POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
05627
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
05628
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<673, 5> { using ZPZ = aerobus::zpz<673>; using type =
05629
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<668»; }; // NOLINT
               template<> struct ConwayPolynomial<673, 6> { using ZPZ = aerobus::zpz<673>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<524>, ZPZV<248>, ZPZV<35>, ZPZV<5»; }; // NOLINT
05631
               template<> struct ConwayPolynomial<673, 7> { using ZPZ = aerobus::zpz<673>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<668»; }; // NOLINT
05632
              template<> struct ConwayPolynomial<673, 8> { using ZPZ = aerobus::zpz<673>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<587>, ZPZV<302>, ZPZV<5»; }; //
         NOLINT
               template<> struct ConwayPolynomial<673, 9> { using ZPZ = aerobus::zpz<673>; using type =
05633
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<347>, ZPZV<553>, ZPZV<668»;
         }; // NOLINT
05634
               template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
        POLYV<ZPZV<1>, ZPZV<675»; }; // NOLINT
               template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
        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 template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
05637
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<677, 5> { using ZPZ = aerobus::zpz<677>; using type =
05638
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<675»; }; // NOLINT
05639
               template<> struct ConwayPolynomial<677, 6> { using ZPZ = aerobus::zpz<677>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446>, ZPZV<632>, ZPZV<50>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type =
05640
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<675»; }; // NOLINT
              template<> struct ConwayPolynomial<677, 8> { using ZPZ = aerobus::zpz<677>; using type =
05641
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3619>, ZPZV<152>, ZPZV<152>, ZPZV<2»; }; //
05642
              template<> struct ConwayPolynomial<677, 9> { using ZPZ = aerobus::zpz<677>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<504>, ZPZV<404>, ZPZV<675»;
         }; // NOLINT
               template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
05643
        POLYV<ZPZV<1>, ZPZV<678»; }; // NOLINT
              template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
        POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; }; // NOLINT
05645
               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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5»; }; // NOLINT
05646
               template<> struct ConwayPolynomial<683, 5> { using ZPZ = aerobus::zpz<683>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<678»; }; // NOLINT
05648
              template<> struct ConwayPolynomial<683, 6> { using ZPZ = aerobus::zpz<683>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<644>, ZPZV<109>, ZPZV<434>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<683, 7> { using ZPZ = aerobus::zpz<683>; using type
05649
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, Z
               template<> struct ConwayPolynomial<683, 8> { using ZPZ = aerobus::zpz<683>; using type
         POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<383>, 2PZV<1844, 2PZV<65>, 2PZV<5»; };
        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>;
05651
         }; // NOLINT
               template<> struct ConwayPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
        POLYV<ZPZV<1>, ZPZV<688»; }; // NOLINT
05653
               template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
        POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
05654
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<688»; }; // NOLINT template<> struct ConwayPolynomial<691, 4> { using ZPZ = aerobus::zpz<691>; using type =
05655
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; }; // NOLINT
05656
              template<> struct ConwayPolynomial<691, 5> { using ZPZ = aerobus::zpz<691>; using type =
        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</pre>
05657
```

```
template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05659
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<356>, ZPZV<425>, ZPZV<321>, ZPZV<3»; }; //
           NOLINT
                 template<> struct ConwayPolynomial<691, 9> { using ZPZ = aerobus::zpz<691>; using type =
05660
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<556>, ZPZV<4433, ZPZV<688»;
           }; // NOLINT
                 template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
05661
          POLYV<ZPZV<1>, ZPZV<699»; }; // NOLINT
                  template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
05662
          POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; }; // NOLINT
05663
                  template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; }; // NOLINT
05664
                 template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
05665
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699»; }; // NOLINT
                  template<> struct ConwayPolynomial<701, 6> { using ZPZ = aerobus::zpz<701>; using type =
05666
          POLYV<2PZV<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 =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<699»; };
05668
                 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<2w; }; //
           NOLINT
                  template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<459>, ZPZV<459>, ZPZV<373>, ZPZV<699»;
           }; // NOLINT
05670
                  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 =
05671
          POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
05672
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707»; }; // NOLINT template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
05673
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<709, 5> { using ZPZ = aerobus::zpz<709>; using type =
05674
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<707»; }; // NOLINT
05675
                  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 =
05676
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<707»; }; // NOLINT template<> struct ConwayPolynomial<709, 8> { using ZPZ = aerobus::zpz<709>; using type =
05677
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<689>, ZPZV<233>, ZPZV<79>, ZPZV<2»; }; //
05678
                  template<> struct ConwayPolynomial<709, 9> { using ZPZ = aerobus::zpz<709>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<257>, ZPZV<171>, ZPZV<707»;
           }; // NOLINT
05679
                  template<> struct ConwavPolynomial<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 =
05680
          POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; }; // NOLINT
05681
                 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 =
05682
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<5>, ZPZV<602>, ZPZV<11»; }; // NOLINT 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 =
05684
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<533>, ZPZV<591>, ZPZV<182>, ZPZV<11»; }; // NOLINT
05685
                 template<> struct ConwayPolynomial<719, 7> { using ZPZ = aerobus::zpz<719>; using type =
          POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708*; }; // NOLINT
05686
                 template<> struct ConwayPolynomial<719, 8> { using ZPZ = aerobus::zpz<719>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<714>, ZPZV<362>, ZPZV<244>, ZPZV<11»; }; //
          template<> struct ConwayPolynomial<719, 9> { using ZPZ = aerobus::zpz<719>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<288>, ZPZV<260>, ZPZV<708»;
05687
           }; // NOLINT
                  template<> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
05688
          POLYV<ZPZV<1>, ZPZV<722»; }; // NOLINT
                 template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
05689
          POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
05690
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<7>, ZPZV<7>; }; // NOLINT template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
05691
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5»; }; // NOLINT
                 template<> struct ConwayPolynomial<727, 5> { using ZPZ = aerobus::zpz<727>; using type =
05692
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; PZV<0>, ZPZV<0>; ZPZV<0>
05693
05694
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<12»; }; // NOLINT
                 template<> struct ConwayPolynomial<727, 8> { using ZPZ = aerobus::zpz<727>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<639>, ZPZV<671>, ZPZV<368>, ZPZV<36»; }; //
          template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<573>, ZPZV<502>, ZPZV<722»;</pre>
05696
```

```
}; // NOLINT
05697
               template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
        POLYV<ZPZV<1>, ZPZV<727»; }; // NOLINT
               template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
05698
        POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; }; // NOLINT
               template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
05699
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727%; }; // NOLINT template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
05700
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<733, 5> { using ZPZ = aerobus::zpz<733>; using type =
05701
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<8>, ZPZV<8>; }; // NOLINT template<> struct ConwayPolynomial<733, 6> { using ZPZ = aerobus::zpz<733>; using type =
05702
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<549>, ZPZV<54>, ZPZV<551>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<733, 7> { using ZPZ = aerobus::zpz<733>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<727»; }; // NOLINT
05704
              template<> struct ConwayPolynomial<733, 8> { using ZPZ = aerobus::zpz<733>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<532>, ZPZV<610>, ZPZV<142>, ZPZV<6»; }; //
        NOLINT
               template<> struct ConwayPolynomial<733, 9> { using ZPZ = aerobus::zpz<733>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<337>, ZPZV<6>, ZPZV<727»; };
05706
               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 =
05707
        POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; }; // NOLINT template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
05709
        template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
05710
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<736»; }; // NOLINT
05711
               template<> struct ConwayPolynomial<739, 6> { using ZPZ = aerobus::zpz<739>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<422>, ZPZV<4447>, ZPZV<625>, ZPZV<3»; }; // NOLINT
05712
              template<> struct ConwayPolynomial<739, 7> { using ZPZ = aerobus::zpz<739>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<44>, ZPZV<736»; }; // NOLINT template<> struct ConwayPolynomial<739, 8> { using ZPZ = aerobus::zpz<739>; using type =
05713
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<401>, ZPZV<169>, ZPZV<25>, ZPZV<3»; }; //
05714
               template<> struct ConwayPolynomial<739, 9> { using ZPZ = aerobus::zpz<739>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<616>, ZPZV<81>, ZPZV<83*;
         }; // NOLINT
05715
               template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<738»; }; // NOLINT
              template<> struct ConwayPolynomial<743, 2> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; }; // NOLINT
05717
               template<> struct ConwayPolynomial<743, 3> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3
, ZPZV<3, ZPZV<3, ZPZV<3, ZPZV<3
, ZPZV<3, ZPZV<3, ZPZV<3
, ZPZV<3, ZPZV<3, ZPZV<3
, ZPZV<3, ZPZV<3
, ZPZV<3, ZPZV<3
, ZPZV<3
05718
05719
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT
               template<> struct ConwayPolynomial<743, 6> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<236>, ZPZV<471>, ZPZV<88>, ZPZV<5»; }; // NOLINT
05721
              template<> struct ConwayPolynomial<743, 7> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<738»; }; // NOLINT
               template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
05722
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<55), ZPZV<551>, ZPZV<279>, ZPZV<588>, ZPZV<5»; }; //
        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»;
         }; // NOLINT
               template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
05724
        POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
               template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
        POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
05726
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; }; // NOLINT
template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<5>, ZPZV<3»; }; // NOLINT
05727
               template<> struct ConwayPolynomial<751, 5> { using ZPZ = aerobus::zpz<751>; using type =
05728
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT
05729
              template<> struct ConwayPolynomial<751, 6> { using ZPZ = aerobus::zpz<751>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<298>, ZPZV<633>, ZPZV<539>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<751, 7> { using ZPZ = aerobus::zpz<751>; using type =
05730
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<748»; };
                                                                                                                                      // NOLINT
               template<> struct ConwayPolynomial<751, 8> { using ZPZ = aerobus::zpz<751>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<741>, ZPZV<243>, ZPZV<672>, ZPZV<3»; };
05732
              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<0>, ZPZV<10>, ZPZV<703>, ZPZV<489>, ZPZV<7489;
         }; // NOLINT
               template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
        POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
05734
              template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
        POLYV<ZPZV<1>, ZPZV<755>, 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
05735
```

```
05736
           template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<757, 5> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<755»; }; // NOLINT 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
05738
           template<> struct ConwayPolynomial<757, 7> { using ZPZ = aerobus::zpz<757>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<755»; };
05740
          template<> struct ConwayPolynomial<757, 8> { using ZPZ = aerobus::zpz<757>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<110>, ZPZV<509>, ZPZV<2»; }; //
       NOLINT
05741
           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<0>, ZPZV<8>, ZPZV<88>, ZPZV<688>, ZPZV<702>, ZPZV<755»;
       }; // NOLINT
05742
           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 =
05743
      POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; }; // NOLINT
           template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755»; }; // NOLINT
           template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type =
05745
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<658>, ZPZV<6»; }; // NOLINT
05746
           template<> struct ConwayPolynomial<761, 5> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT
05747
           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 =
05748
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<761, 8> { using ZPZ = aerobus::zpz<761>; using type =
05749
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<603>, ZPZV<144>, ZPZV<540>, ZPZV<5w; }; //
       NOLINT
05750
           template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<317>, ZPZV<571>, ZPZV<755»;
       }; // NOLINT
05751
           template<> struct ConwayPolynomial<769, 1> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<758»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 2> { using ZPZ = aerobus::zpz<769>; using type =
05752
      POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; }; // NOLINT
05753
           template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; }; // NOLINT template<> struct ConwayPolynomial<769, 4> { using ZPZ = aerobus::zpz<769>; using type =
05754
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<769, 5> { using ZPZ = aerobus::zpz<769>; using type =
05755
      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
05757
           template<> struct ConwayPolynomial<769, 7> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<758»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 8> { using ZPZ = aerobus::zpz<769>; using type =
05758
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<560>, ZPZV<574>, ZPZV<632>, ZPZV<11»; }; //
           template<> struct ConwayPolynomial<769, 9> { using ZPZ = aerobus::zpz<769>; using type =
05759
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<623>, ZPZV<751>, ZPZV<758»;
       }; // NOLINT
05760
           template<> struct ConwayPolynomial<773, 1> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<771»; }; // NOLINT
           template<> struct ConwayPolynomial<773, 2> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
05762
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<771»; }; // NOLINT
template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<773, 5> { using ZPZ = aerobus::zpz<773>; using type =
05763
05764
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<771»; }; // NOLINT
05765
           template<> struct ConwayPolynomial<773, 6> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9), ZPZV<91>, ZPZV<581>, ZPZV<581>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<773, 7> { using ZPZ = aerobus::zpz<773>; using type =
05766
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<771»; }; // NOLINT
           template<> struct ConwayPolynomial<773, 8> { using ZPZ = aerobus::zpz<773>; using type =
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<484>, ZPZV<94>, ZPZV<693>, ZPZV<2»; };
05768
          template<> struct ConwayPolynomial<773, 9> { using ZPZ = aerobus::zpz<773>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<216>, ZPZV<574>, ZPZV<771»;
       }; // NOLINT
05769
           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 =
05770
      POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
05771
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785»; }; // NOLINT
template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2»; }; // NOLINT
05772
           template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<785»; }; // NOLINT
05774
          template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<2P2V<1>, ZPZV<0>, ZPZV<0>, ZPZV<98>, ZPZV<512>, ZPZV<606>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<787, 7> { using ZPZ = aerobus::zpz<787>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<785»; };
                   template<> struct ConwayPolynomial<787, 8> { using ZPZ = aerobus::zpz<787>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<612>, ZPZV<26>, ZPZV<715>, ZPZV<2»; }; //
           NOLINT
05777
                   template<> struct ConwayPolynomial<787, 9> { using ZPZ = aerobus::zpz<787>; using type =
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<480>, ZPZV<573>, ZPZV<785»;
           }; // NOLINT
05778
                   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 =
05779
           POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
05780
           POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type =
05781
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<717>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<797, 5> { using ZPZ = aerobus::zpz<797>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<795»; }; // NOLINT</pre>
05782
                   template<> struct ConwayPolynomial<797, 6> { using ZPZ = aerobus::zpz<797>; using type =
05783
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<657>, ZPZV<396>, ZPZV<71>, ZPZV<2»; }; // NOLINT
05784
                  template<> struct ConwayPolynomial<797,
                                                                                               7> { using ZPZ = aerobus::zpz<797>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<795»; }; //
05785
                  template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2»; }; //
           NOLINT
05786
                   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<240>, ZPZV<599>, ZPZV<795»;
           }; // NOLINT
05787
                  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 =
05788
           POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
05789
                   template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<806»; }; // NOLINT template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
05790
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<809, 5> { using ZPZ = aerobus::zpz<809>; using type =
05791
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; }; // NOLINT
                   template<> struct ConwayPolynomial<809, 6> { using ZPZ = aerobus::zpz<809>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<562>, ZPZV<75>, ZPZV<43>, ZPZV<3»; };
                  template<> struct ConwayPolynomial<809, 7> { using ZPZ = aerobus::zpz<809>; using type =
05793
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<806»; }; // NOLINT
                  template<> struct ConwayPolynomial<809, 8> { using ZPZ = aerobus::zpz<809>; using type =
05794
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<593>, ZPZV<745>, ZPZV<673>, ZPZV<673»; }; //
                   template<> struct ConwayPolynomial<809, 9> { using ZPZ = aerobus::zpz<809>; using type
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<341>, ZPZV<727>, ZPZV<806»;
           }; // NOLINT
05796
                   template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
           POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
                   template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
05797
           POLYV<ZPZV<1>, ZPZV<806>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<811, 3> { using ZPZ = aerobus::zpz<811>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<808»; }; // NOLINT template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type =
05799
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<811, 5> { using ZPZ = aerobus::zpz<811>; using type =
05800
           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 =
05801
           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 =
05802
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<808»; }; // NOLINT template<> struct ConwayPolynomial<811, 8> { using ZPZ = aerobus::zpz<811>; using type =
05803
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<5.
05804
                  template<> struct ConwayPolynomial<811, 9> { using ZPZ = aerobus::zpz<811>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<382>, ZPZV<200>, ZPZV<808»;
            }; // NOLINT
05805
                   \texttt{template<> struct ConwayPolynomial<821, 1> \{ using \ ZPZ = aerobus:: zpz<821>; using \ type = aerobus:: zpz<821>; usi
           POLYV<ZPZV<1>, ZPZV<819»; }; // NOLINT
                   template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
           POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT
05807
                  template<> struct ConwayPolynomial<821, 3> { using ZPZ = aerobus::zpz<821>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<819»; }; // NOLINT template<> struct ConwayPolynomial<821, 4> { using ZPZ = aerobus::zpz<821>; using type =
05808
           POLYY<ZPZY<1>, ZPZV<0>, ZPZV<15>, ZPZV<662>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<819»; }; // NOLINT
05810
                   template<> struct ConwayPolynomial<821, 6> { using ZPZ = aerobus::zpz<821>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<160>, ZPZV<130>, ZPZV<803>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type :
05811
           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 =
05812
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<626>, ZPZV<556>, ZPZV<589>, ZPZV<2»; };
           NOLINT
05813
                  \texttt{template<>} \texttt{struct ConwayPolynomial<821, 9> \{ \texttt{using ZPZ = aerobus::zpz<821>; using type = 2000 \texttt{monormial} \texttt{struct} = 2000 \texttt{monormial} = 2000 \texttt{m
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<650>, ZPZV<557>, ZPZV<819»;
           }; // NOLINT
```

```
template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
05815
           template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; }; // NOLINT
05816
           template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT
05818
           template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>; }; // NOLINT
           template<> struct ConwayPolynomial<823, 6> { using ZPZ = aerobus::zpz<823>; using type =
05819
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<822>, ZPZV<6165, ZPZV<744>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type
05820
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<820»; }; //
          template<> struct ConwayPolynomial<823, 8> { using ZPZ = aerobus::zpz<823>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<451>, ZPZV<437>, ZPZV<31>, ZPZV<3»; };
       NOLTNT
      template<> struct ConwayPolynomial<823, 9> { using ZPZ = aerobus::zpz<823>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<740>, ZPZV<609>, ZPZV<820»;
05822
           template<> struct ConwayPolynomial<827, 1> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<825»; }; // NOLINT
           template<> struct ConwayPolynomial<827, 2> { using ZPZ = aerobus::zpz<827>; using type =
05824
      POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
05825
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825»; }; // NOLINT
           template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type =
05826
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<827, 5> { using ZPZ = aerobus::zpz<827>; using type =
05827
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<825*; }; // NOLINT 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
05828
           template<> struct ConwayPolynomial<827, 7> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<825»; }; // NOLINT
05830
           template<> struct ConwayPolynomial<827, 8> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<812>, ZPZV<79>, ZPZV<32>, ZPZV<2»; };
       NOLINT
           template<> struct ConwayPolynomial<827, 9> { using ZPZ = aerobus::zpz<827>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<372>, ZPZV<825»;
       }; // NOLINT
05832
           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 =
05833
      POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<827»; }; // NOLINT template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
05835
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<604>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<829, 5> { using ZPZ = aerobus::zpz<829>; using type =
05836
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<827»; }; // NOLINT
           template<> struct ConwayPolynomial<829, 6> { using ZPZ = aerobus::zpz<829>; using type =
05837
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<476>, ZPZV<817>, ZPZV<2»; }; // NOLIN
05838
           template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<827»; }; // NOLINT template<> struct ConwayPolynomial<829, 8> { using ZPZ = aerobus::zpz<829>; using type =
05839
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<468>, ZPZV<241>, ZPZV<138>, ZPZV<2»; }; //
05840
           template<> struct ConwayPolynomial<829, 9> { using ZPZ = aerobus::zpz<829>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<621>, ZPZV<552>, ZPZV<827»;
       }; // NOLINT
05841
           template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<828»; }; // NOLINT
05842
            template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
       POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; }; // NOLINT
05843
           template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; ); // NOLINT
template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11»; }; // NOLINT
template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
05844
05845
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<828»; }; // NOLINT
           template<> struct ConwayPolynomial<839, 6> { using ZPZ = aerobus::zpz<839>; using type =
05846
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<370>, ZPZV<537>, ZPZV<23>, ZPZV<11»; }; // NOLINT
05847
           template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<828»; }; // NOLINT template<> struct ConwayPolynomial<839, 8> { using ZPZ = aerobus::zpz<839>; using type =
05848
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<553>, ZPZV<779>, ZPZV<329>, ZPZV<11»; }; //
       NOLINT
05849
           template<> struct ConwayPolynomial<839, 9> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<349>, ZPZV<206>, ZPZV<828»;
       }; // NOLINT
05850
            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 =
      POLYV<ZPZV<1>, ZPZV<852>, ZPZV<2»; }; // NOLINT
05852
          template<> struct ConwayPolynomial<853, 3> { using ZPZ = aerobus::zpz<853>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<2»; };
               template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<851»; // NOLINT
               template<> struct ConwayPolynomial<853, 6> { using ZPZ = aerobus::zpz<853>; using type =
05855
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<276>, ZPZV<1944, ZPZV<512>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type
05856
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<851»; }; //
               template<> struct ConwayPolynomial<853, 8> { using ZPZ = aerobus::zpz<853>; using type
05857
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<544>, ZPZV<846>, ZPZV<118>, ZPZV<2»; }; //
         template<> struct ConwayPolynomial<853, 9> { using ZPZ = aerobus::zpz<853>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<851»;
05858
         }; // NOLINT
  template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
         POLYV<ZPZV<1>, ZPZV<854»; }; // NOLINT
05860
              template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
         POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
05861
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; }; // NOLINT
               template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
05862
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<528>, ZPZV<3»; }; // NOLINT
05863
               template<> struct ConwayPolynomial<857, 5> { using ZPZ = aerobus::zpz<857>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<854»; }; // NOLINT template<> struct ConwayPolynomial<857, 6> { using ZPZ = aerobus::zpz<857>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<32>, ZPZV<824>, ZPZV<65>, ZPZV<3»; }; // NOLINT
05864
               template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::zpz<857>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, Z
              template<> struct ConwayPolynomial<857, 8> { using ZPZ = aerobus::zpz<857>; using type =
05866
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<611>, ZPZV<552>, ZPZV<494>, ZPZV<3»; }; //
         NOLINT
              template<> struct ConwayPolynomial<857, 9> { using ZPZ = aerobus::zpz<857>; using type =
05867
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<719>, ZPZV<854»;
         }; // NOLINT
               template<> struct ConwayPolynomial<859, 1> { using ZPZ = aerobus::zpz<859>; using type =
05868
         POLYV<ZPZV<1>, ZPZV<857»; }; // NOLINT
               template<> struct ConwayPolynomial<859, 2> { using ZPZ = aerobus::zpz<859>; using type =
05869
         POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type =
05871
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<859, 5> { using ZPZ = aerobus::zpz<859>; using type =
05872
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<857»; }; // NOLINT
05873
               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
05874
               template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpz<859>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 8> { using ZPZ = aerobus::zpz<859>; using type =
05875
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<446>, ZPZV<672>, ZPZV<672>; }; //
         NOLINT
05876
               template<> struct ConwayPolynomial<859, 9> { using ZPZ = aerobus::zpz<859>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<648>, ZPZV<845>, ZPZV<857»;
         }; // NOLINT
05877
               template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
         POLYV<ZPZV<1>, ZPZV<858»; }; // NOLINT
               template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
05878
         POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
05879
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<588%; }; // NOLINT template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
05880
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<863, 5> { using ZPZ = aerobus::zpz<863>; using type =
05881
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<858»; }; // NOLINT
               template<> struct ConwayPolynomial<863, 6> { using ZPZ = aerobus::zpz<863>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<330>, ZPZV<62>, ZPZV<300>, ZPZV<5»; }; // NOLINT
05883
              template<> struct ConwayPolynomial<863, 7> { using ZPZ = aerobus::zpz<863>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<858»; }; // NOLINT template<> struct ConwayPolynomial<863, 8> { using ZPZ = aerobus::zpz<863>; using type =
05884
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<576>, ZPZV<576>, ZPZV<849>, ZPZV<59»; }; //
         NOLINT
               template<> struct ConwayPolynomial<863, 9> { using ZPZ = aerobus::zpz<863>; using type =
05885
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<381>, ZPZV<31>, ZPZV<1>, ZPZV<858»; };
          // NOLINT
05886
               template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
         POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
         POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; }; // NOLINT
05888
               template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875»; }; // NOLINT template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type =
05889
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<877, 5> { using ZPZ = aerobus::zpz<877>; using type =
05890
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<875»; }; // NOLINT
05891
              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 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»; }; // NOLINT
05892
```

```
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<34>, ZPZV<34>, ZPZV<319>, ZPZV<347>, ZPZV<347</p>
        NOLINT
        template<> struct ConwayPolynomial<877, 9> { using ZPZ = aerobus::zpz<877>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<770>, ZPZV<278>, ZPZV<875»;</pre>
05894
        }; // NOLINT
05895
               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 =
05896
        POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
05897
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<1>, ZPZV<878»; }; // NOLINT template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
05898
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4447>, ZPZV<3»; }; // NOLINT
05899
             template<> struct ConwayPolynomial<881, 5> { using ZPZ = aerobus::zpz<881>; using type =
        template<> struct ConwayPolynomial<881, 6> { using ZPZ = aerobus::zpz<881>; using type =
05900
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<18>, ZPZV<218>, ZPZV<231>, ZPZV<3); / NoLINT template<> struct ConwayPolynomial<881, 7> { using ZPZ = aerobus::zpz<881>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<878»; };
              template<> struct ConwayPolynomial<881, 8> { using ZPZ = aerobus::zpz<881>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<635>, ZPZV<490>, ZPZV<561>, ZPZV<3»; }; //
        template<> struct ConwayPolynomial<881, 9> { using ZPZ = aerobus::zpz<881>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<587>, ZPZV<510>, ZPZV<878»;</pre>
05903
        }; // NOLINT
             template<> struct ConwayPolynomial<883, 1> { using ZPZ = aerobus::zpz<883>; using type =
05904
        POLYV<ZPZV<1>, ZPZV<881»; }; // NOLINT
05905
              template<> struct ConwayPolynomial<883, 2> { using ZPZ = aerobus::zpz<883>; using type =
        POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<883, 3> { using ZPZ = aerobus::zpz<883>; using type =
05906
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<881»; }; // NOLINT
              template<> struct ConwayPolynomial<883, 4> { using ZPZ = aerobus::zpz<883>; using type =
05907
        05908
              template<> struct ConwayPolynomial<883, 5> { using ZPZ = aerobus::zpz<883>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<881»; }; // NOLINT
        template<> struct ConwayPolynomial<883, 6> { using ZPZ = aerobus::zpz<883>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<879>, ZPZV<865>, ZPZV<871>, ZPZV<2»; }; // NOLINT
05909
05910
              template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<881»; };
05911
             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<28; }; //
        NOLINT
05912
              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<5>, ZPZV<360>, ZPZV<557>, ZPZV<881»;
        }; // NOLINT
05913
              template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
        POLYV<ZPZV<1>, ZPZV<882»; }; // NOLINT
              template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
05914
        POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
05915
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<882»; }; // NOLINT
05916
              template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<887, 5> { using ZPZ = aerobus::zpz<887>; using type =
05917
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<882»; }; // NOLINT
              template<> struct ConwayPolynomial<887, 6> { using ZPZ = aerobus::zpz<887>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<775>, ZPZV<341>, ZPZV<28>, ZPZV<5»; }; // NOLINI
              template<> struct ConwayPolynomial<887, 7> { using ZPZ = aerobus::zpz<887>; using type =
05919
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<882»; }; // NOLIN template<> struct ConwayPolynomial<887, 8> { using ZPZ = aerobus::zpz<887>; using type =
05920
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<781>, ZPZV<381>, ZPZV<706>, ZPZV<5»; }; //
        NOLINT
              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
05922
              template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
        POLYV<ZPZV<1>, ZPZV<905»; }; // NOLINT
              template<> struct ConwayPolynomial<907, 2> { using ZPZ = aerobus::zpz<907>; using type =
05923
        POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<907, 3> { using ZPZ = aerobus::zpz<907>; using type =
05924
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<905»; }; // NOLINT template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
05925
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
05926
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; }; // NOLINT
              template<> struct ConwayPolynomial<907, 6> { using ZPZ = aerobus::zpz<907>; using type =
05927
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<752>, ZPZV<266>, ZPZV<2%; }; // NOLINT template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type =
05928
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; }; // NOLINT
              template<> struct ConwayPolynomial<907, 8> { using ZPZ = aerobus::zpz<907>; using type
05929
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<584>, ZPZV<518>, ZPZV<811>, ZPZV<2»; }; //
05930
             template<> struct ConwayPolynomial<907, 9> { using ZPZ = aerobus::zpz<907>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<783>, ZPZV<57>, ZPZV<905»;
        }; // NOLINT
05931
             template<> struct ConwayPolynomial<911, 1> { using ZPZ = aerobus::zpz<911>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<894»; };
               template<> struct ConwayPolynomial<911, 2> { using ZPZ = aerobus::zpz<911>; using type =
        POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; }; // NOLINT
              template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
05933
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; }; // NOLINT template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
05934
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17»; }; // NOLINT
               template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
05935
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<894»; }; // NOLINT
05936
              template<> struct ConwayPolynomial<911, 6> { using ZPZ = aerobus::zpz<911>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<683>, ZPZV<19>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type = DOLYWARDEN(4)> SPZV<60 | SPZV<60
05937
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<49, ZPZV<894»; };
               template<> struct ConwayPolynomial<911, 8> { using ZPZ = aerobus::zpz<911>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<5>, ZPZV<590>, ZPZV<168>, ZPZV<110*; }; //
              template<> struct ConwayPolynomial<911, 9> { using ZPZ = aerobus::zpz<911>; using type =
05939
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<679>, ZPZV<679>, ZPZV<6165, ZPZV<894»;
         }; // NOLINT
               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 =
05941
        POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; }; // NOLINT
              template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
05942
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912*; }; // NOLINT template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7»; }; // NOLINT
05944
              template<> struct ConwayPolynomial<919, 5> { using ZPZ = aerobus::zpz<919>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<912»; }; // NOLINT
05945
              template<> struct ConwayPolynomial<919, 6> { using ZPZ = aerobus::zpz<919>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<312>, ZPZV<312>, ZPZV<113>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<919, 7> { using ZPZ = aerobus::zpz<919>; using type
05946
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<9>, ZPZV<9>, ZPZV<9
05947
             template<> struct ConwayPolynomial<919, 8> { using ZPZ = aerobus::zpz<919>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<202>, ZPZV<504>, ZPZV<7»; }; //
         NOLINT
        template<> struct ConwayPolynomial<919, 9> { using ZPZ = aerobus::zpz<919>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<410>, ZPZV<623>, ZPZV<912»;
05948
         }; // NOLINT
05949
               template<> struct ConwayPolynomial<929, 1> { using ZPZ = aerobus::zpz<929>; using type =
        POLYV<ZPZV<1>, ZPZV<926»; }; // NOLINT
              template<> struct ConwayPolynomial<929, 2> { using ZPZ = aerobus::zpz<929>; using type =
05950
        POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3»: }: // NOLINT
05951
              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 =
05952
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
05953
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<926»; }; // NOLINT
              template<> struct ConwayPolynomial<929, 6> { using ZPZ = aerobus::zpz<929>; using type =
05954
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<805>, ZPZV<92>, ZPZV<86>, ZPZV<3»; };
              template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<926»; }; // NOLINT
05956
              template<> struct ConwayPolynomial<929, 8> { using ZPZ = aerobus::zpz<929>; using type =
         POLYV<2PZV<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<0>, ZPZV<0>, ZPZV<481>, ZPZV<199>, ZPZV<926»;
         }; // NOLINT
05958
               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 =
05959
        POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<937, 3> { using ZPZ = aerobus::zpz<937>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<932»; }; // NOLINT template<> struct ConwayPolynomial<937, 4> { using ZPZ = aerobus::zpz<937>; using type =
05961
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<23>, ZPZV<585>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
05962
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5932»; }; // NOLINT
05963
               template<> struct ConwayPolynomial<937, 6> { using ZPZ = aerobus::zpz<937>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<794>, ZPZV<727>, ZPZV<934>, ZPZV<5»; }; // NOLINT
05964
              template<> struct ConwayPolynomial<937, 7> { using ZPZ = aerobus::zpz<937>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2, ZPZV<2, ZPZV<2, ZPZV<2, ZPZV<23; }; // NOLINT template<> struct ConwayPolynomial<937, 8> { using ZPZ = aerobus::zpz<937>; using type =
05965
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<26>, ZPZV<53>, ZPZV<5»; };
              template<> struct ConwayPolynomial<937, 9> { using ZPZ = aerobus::zpz<937>; using type =
05966
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<533>, ZPZV<483>, ZPZV<932»;
         }; // NOLINT
05967
              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 =
        POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2»; }; // NOLINT
05969
              template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2»; }; // NOLINT
05970
```

```
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
05972
                        template<> struct ConwayPolynomial<941, 6> { using ZPZ = aerobus::zpz<941>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<45>, ZPZV<694>, ZPZV<538>, ZPZV<2x; }; // NOLINT template<> struct ConwayPolynomial<941, 7> { using ZPZ = aerobus::zpz<941>; using type
05973
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<939»; };
                                                                                                                                                                                                                                // NOLINT
                         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
              template<> struct ConwayPolynomial<941, 9> { using ZPZ = aerobus::zpz<941>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<70>, ZPZV<708>, ZPZV<197>, ZPZV<939»;</pre>
05975
               }; // NOLINT
05976
                         template<> struct ConwayPolynomial<947, 1> { using ZPZ = aerobus::zpz<947>; using type =
              POLYV<ZPZV<1>, ZPZV<945»; }; // NOLINT
05977
                         template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
              POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
05978
              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
                          template<> struct ConwayPolynomial<947, 5> { using ZPZ = aerobus::zpz<947>; using type =
05980
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<945»; }; // NOLINT
                        template<> struct ConwayPolynomial<947, 6> { using ZPZ = aerobus::zpz<947>; using type =
05981
              POLYY<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<880>, ZPZV<7>, ZPZV<95>, ZPZV<95>, ZPZV<2>; ; // NOLINT template<> struct ConwayPolynomial<947, 7> { using ZPZ = aerobus::zpz<947>; using type
05982
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 >, ZPZV
05983
                        template<> struct ConwayPolynomial<947, 8> { using ZPZ = aerobus::zpz<947>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<845>, ZPZV<597>, ZPZV<581>, ZPZV<2»; }; //
               NOLINT
05984
                        template<> struct ConwayPolynomial<947, 9> { using ZPZ = aerobus::zpz<947>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<269>, ZPZV<808>, ZPZV<945»;
               }; // NOLINT
                         template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
05985
               POLYV<ZPZV<1>, ZPZV<950»; }; // NOLINT
                         template<> struct ConwayPolynomial<953, 2> { using ZPZ = aerobus::zpz<953>; using type =
05986
              POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3»; }; // NOLINT
                         template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
05987
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950»; }; // NOLINT
05988
                         template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
              POLYY<ZPZY<1>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<3%; }; // NOLINT template<> struct ConwayPolynomial<953, 5> { using ZPZ = aerobus::zpz<953>; using type =
05989
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<5>, ZPZV<50>; }; // NOLINT template<> struct ConwayPolynomial<953, 6> { using ZPZ = aerobus::zpz<953>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<507>, ZPZV<829>, ZPZV<730>, ZPZV<3»; }; // NOLINT
05990
                         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
05992
                       template<> struct ConwayPolynomial<953, 8> { using ZPZ = aerobus::zpz<953>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<579>, ZPZV<658>, ZPZV<108>, ZPZV<3»; }; //
               NOLINT
                         template<> struct ConwayPolynomial<953, 9> { using ZPZ = aerobus::zpz<953>; using type =
05993
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<819>, ZPZV<316>, ZPZV<950»;
               }; // NOLINT
05994
                         template<> struct ConwayPolynomial<967, 1> { using ZPZ = aerobus::zpz<967>; using type =
              POLYV<ZPZV<1>, ZPZV<962»; }; // NOLINT
                        template<> struct ConwayPolynomial<967, 2> { using ZPZ = aerobus::zpz<967>; using type =
05995
              POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5»; }; // NOLINT
                         template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
05997
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; }; // NOLINT
                        template<> struct ConwayPolynomial<967, 5> { using ZPZ = aerobus::zpz<967>; using type =
05998
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<962»; }; // NOLINT
05999
                         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<83; }; // NOLINT
06000
                       template<> struct ConwayPolynomial<967, 7> { using ZPZ = aerobus::zpz<967>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 8> { using ZPZ = aerobus::zpz<967>; using type =
06001
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<840>, ZPZV<502>, ZPZV<136>, ZPZV<5»; }; //
               NOLINT
                         template<> struct ConwayPolynomial<967, 9> { using ZPZ = aerobus::zpz<967>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<512>, ZPZV<783>, ZPZV<962»;
               }; // NOLINT
06003
                         template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
              POLYV<ZPZV<1>, ZPZV<965»; }; // NOLINT
                         template<> struct ConwayPolynomial<971, 2> { using ZPZ = aerobus::zpz<971>; using type =
06004
               POLYV<ZPZV<1>, ZPZV<970>, ZPZV<6»; }; // NOLINT
                        template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
06005
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<965»; }; // NOLINT template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
06006
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<527>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
06007
              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 =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<970>, ZPZV<729>, ZPZV<718>, ZPZV<6»; }; // NOLINT
06009
                       template<> struct ConwayPolynomial<971, 7> { using ZPZ = aerobus::zpz<971>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<725>, ZPZV<281>, ZPZV<206>, ZPZV<6*; }; //
06011
               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<805>, ZPZV<805>, ZPZV<805»;
         }; // NOLINT
06012
                template<> struct ConwayPolynomial<977, 1> { using ZPZ = aerobus::zpz<977>; using type =
         POLYV<ZPZV<1>, ZPZV<974»; }; // NOLINT
06013
                template<> struct ConwayPolynomial<977, 2> { using ZPZ = aerobus::zpz<977>; using type =
         POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; }; // NOLINT
06014
               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 =
06015
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<800>, ZPZV<800>, ZPZV<800>, ZPZV<800>; ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<977, 5> { using ZPZ = aerobus::zpz<977>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<974»; }; // NOLINT
        template<> struct ConwayPolynomial<977, 6> { using ZPZ = aerobus::zpz<977>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<729>, ZPZV<830>, ZPZV<753>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<977, 7> { using ZPZ = aerobus::zpz<977>; using type =
06017
06018
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<974»; }; // NOLINT
06019
               template<> struct ConwayPolynomial<977, 8> { using ZPZ = aerobus::zpz<977>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<855>, ZPZV<807>, ZPZV<77>, ZPZV<3»; };
         NOLINT
               template<> struct ConwayPolynomial<977, 9> { using ZPZ = aerobus::zpz<977>; using type =
06020
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<740>, ZPZV<974»;
         }; // NOLINT
                template<> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
         POLYV<ZPZV<1>, ZPZV<978»; }; // NOLINT
               template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
06022
         POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
06023
         POLYVCZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<2
, ZPZV<2
06024
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5»; }; // NOLINT
06025
              template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
         template<> struct ConwayPolynomial<983, 6> { using ZPZ = aerobus::zpz<983>; using type =
06026
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<849, ZPZV<26>, ZPZV<228>, ZPZV<5»; }; // NOLINT 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<978»; };
               template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
06028
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<5»; }; //
         NOLINT
               template<> struct ConwayPolynomial<983, 9> { using ZPZ = aerobus::zpz<983>; using type =
06029
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<87>, ZPZV<87>, ZPZV<87>, ZPZV<87>, ZPZV<87>, ZPZV<87>, ZPZV<87
06030
               template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
         POLYV<ZPZV<1>, ZPZV<985»; }; // NOLINT
               template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
06031
         POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; }; // NOLINT
               template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
06032
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<955»; }; // NOLINT template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
          \verb"POLYV<ZPZV<1>, \ \verb"ZPZV<0>, \ \verb"ZPZV<10>, \ \verb"ZPZV<794>, \ \verb"ZPZV<6"; \ \verb"}; \ \ // \ \verb"NOLINT" 
        template<> struct ConwayPolynomial<991, 5> { using ZPZ = aerobus::zpz<991>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<985»; }; // NOLINT
06034
               template<> struct ConwayPolynomial<991, 6> { using ZPZ = aerobus::zpz<991>; using type =
06035
         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
06036
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<985»; }; // NOLINT template<> struct ConwayPolynomial<991, 8> { using ZPZ = aerobus::zpz<991>; using type =
06037
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<941>, ZPZV<786>, ZPZV<234>, ZPZV<6»; }; //
         NOLINT
06038
               template<> struct ConwayPolynomial<991, 9> { using ZPZ = aerobus::zpz<991>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<466>, ZPZV<266, ZPZV<285»;
         }; // NOLINT
06039
               template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
         POLYV<ZPZV<1>, ZPZV<990»; }; // NOLINT
               template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
06040
         POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; }; // NOLINT
                template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
06041
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990»; }; // NOLINT template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
06042
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<62>, ZPZV<622>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
06043
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10, ZPZV<90»; }; // NOLINT template<> struct ConwayPolynomial<997, 6> { using ZPZ = aerobus::zpz<997>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<981>, ZPZV<58>, ZPZV<260>, ZPZV<7»; }; // NOLINT
06045
               template<> struct ConwayPolynomial<997, 7> { using ZPZ = aerobus::zpz<997>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<90»; }; // 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<3>, ZPZV<473>, ZPZV<241>, ZPZV<241>, ZPZV<241>; }; //
06046
         template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<732>, ZPZV<616>, ZPZV<990»;
}; // NOLINT
06048 #endif // DO_NOT_DOCUMENT
06049 } // namespace aerobus
```

```
06050 #endif // AEROBUS_CONWAY_IMPORTS
06051
06052 #endif // __INC_AEROBUS__ // NOLINT
```

9.4 src/examples.h File Reference

9.5 examples.h

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 -O3 -std=c++20 -arch=sm_90 fp16.cu
#include <cstdio>
#define WITH CUDA FP16
#include "../src/aerobus.h"
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;
constexpr size_t N = 1 « 24;
template<typename T>
struct Expm1Degree;
template<>
struct Expm1Degree<double> {
    static constexpr size_t val = 18;
template<>
struct Expm1Degree<float> {
   static constexpr size_t val = 11;
template<>
struct ExpmlDegree<__half2> {
    static constexpr size_t val = 6;
double rand(double min, double max) {
  double range = (max - min);
  double div = RAND_MAX / range;
 return min + (rand() / div); // NOLINT
template<typename T>
struct GetRandT;
template<>
struct GetRandT<double> {
    static double func(double min, double max) {
        return rand(min, max);
};
template<>
struct GetRandT<float> {
    static float func(double min, double max) {
  return (float) rand(min, max);
};
template<>
struct GetRandT<__half2> {
    static __half2 func(double min, double max) {
       return __half2(__float2half((float)rand(min, max)), __float2half((float)rand(min, max)));
};
using EXPM1 = aerobus::expm1<int_type, Expm1Degree<float_type>::val>;
 _device__ INLINED float_type f(float_type x) {
    return EXPM1::eval(x);
```

```
}
__global__ void run(size_t N, float_type* in, float_type* out) {
   out[i] = f(f(f(f(f(in[i])))));
int main() {
   float_type *d_in, *d_out;
   float_type *in = reinterpret_cast<float_type*>(malloc(N * sizeof(float_type)));
   float_type *out = reinterpret_cast<float_type*>(malloc(N * sizeof(float_type)));
   for(size_t i = 0; i < N; ++i) \{
      in[i] = GetRandT<float_type>::func(-0.01, 0.01);
   \verb|cudaMemcpy| (d_in, in, N * size of (float_type), cudaMemcpyHostToDevice);|\\
   run«<128, 512»>(N, d_in, d_out);
   cudaMemcpy(out, d_out, N * sizeof(float_type), cudaMemcpyDeviceToHost);
   cudaFree(d_in);
   cudaFree(d_out);
```

10.4 examples/continued_fractions.cpp

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

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;
}
```

208 Examples

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 $a \star n$ 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< 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> 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;

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 1+1+1+1+1+1 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

publication

See also

```
// 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 T exact_large(const T& x) {
     return pow_scalar<T, 5>(0.75 - x) * pow_scalar<T, 11>(1 - x);
template<typename T>
DEVICE 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) {
    ::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>
          using 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	