# Aerobus

v1.2

Generated by Doxygen 1.9.8

1 l	ntroduction	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	7
2 N	lamespace Index	9
	2.1 Namespace List	9
	2	Ū
3 (	Concept Index	11
	3.1 Concepts	11
4 (	Class Index	13
	4.1 Class List	13
5 F	File Index	15
	5.1 File List	15
6 N	lamespace Documentation	17
	6.1 aerobus Namespace Reference	17
	6.1.1 Detailed Description	21
	6.1.2 Typedef Documentation	21
	6.1.2.1 abs_t	21
	6.1.2.2 add_t	22
	6.1.2.3 addfractions_t	22
	6.1.2.4 alternate_t	22
	6.1.2.5 asin	22
	6.1.2.6 asinh	22
	6.1.2.7 atan	23
	6.1.2.8 atanh	23
	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	 24
6.1.2.13 cosh	 24
6.1.2.14 div_t	 24
6.1.2.15 E_fraction	 24
6.1.2.16 embed_int_poly_in_fractions_t	 24
6.1.2.17 exp	 25
6.1.2.18 expm1	 25
6.1.2.19 factorial_t	 25
6.1.2.20 fpq32	 25
6.1.2.21 fpq64	 25
6.1.2.22 FractionField	 25
6.1.2.23 gcd_t	 26
6.1.2.24 geometric_sum	 26
6.1.2.25 lnp1	 26
6.1.2.26 make_frac_polynomial_t	 26
6.1.2.27 make_int_polynomial_t	 27
6.1.2.28 make_q32_t	 27
6.1.2.29 make_q64_t	 27
6.1.2.30 makefraction_t	 27
6.1.2.31 mul_t	 27
6.1.2.32 mulfractions_t	 28
6.1.2.33 pi64	 28
6.1.2.34 PI_fraction	 28
6.1.2.35 pow_t	 28
6.1.2.36 pq64	 28
6.1.2.37 q32	 28
6.1.2.38 q64	 28
6.1.2.39 sin	 29
6.1.2.40 sinh	 29
6.1.2.41 SQRT2_fraction	 29
6.1.2.42 SQRT3_fraction	 29
6.1.2.43 stirling_1_signed_t	 29
6.1.2.44 stirling_1_unsigned_t	 29
6.1.2.45 stirling_2_t	 30
6.1.2.46 sub_t	 30
6.1.2.47 tan	 30
6.1.2.48 tanh	 30
6.1.2.49 taylor	 31
6.1.2.50 vadd_t	 31
6.1.2.51 vmul_t	 31
6.1.3 Function Documentation	 31
6.1.3.1 aligned_malloc()	 31

6.1.4 Variable Documentation       3.         6.1.4.1 alternate_v       3.         6.1.4.2 bernoulli_v       3.         6.1.4.3 combination_v       3.         6.1.4.4 factorial_v       3.         6.2 aerobus::internal Namespace Reference       3.         6.2 aerobus::internal Namespace Reference       3.         6.2.1 Detailed Description       3.         6.2.2 Typedef Documentation       3.         6.2.2.1 make_index_sequence_reverse       3.         6.2.3 Function Documentation       3.         6.2.3.1 index_sequence_reverse()       3.         6.2.4 Variable Documentation       3.         6.2.4.1 is_instantiation_of_v       3.         6.3 aerobus::known_polynomials Namespace Reference       3.         6.3.1 Detailed Description       3.         6.3.2 Enumeration Type Documentation       3.         7 Concept Documentation       3.         7.1.1 Concept definition       3.         7.1.2 Detailed Description       3.         7.2 aerobus::IsField Concept Reference       3.         7.2 aerobus::IsField Concept Reference       3.
6.1.4.2 bernoulli_v
6.1.4.3 combination_v 6.1.4.4 factorial_v 6.2 aerobus::internal Namespace Reference 6.2.1 Detailed Description 6.2.2 Typedef Documentation 6.2.2.1 make_index_sequence_reverse 6.2.2.1 make_index_sequence_reverse 6.2.2.2 type_at_t 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 6.3 aerobus::known_polynomials Namespace Reference 6.3.1 Detailed Description 6.3.2 Enumeration Type Documentation 6.3.2.1 hermite_kind 7 Concept Documentation 7.1 aerobus::lsEuclideanDomain Concept Reference 7.1.1 Concept definition 3.7.1.2 Detailed Description
6.1.4.4 factorial_v       3.         6.2 aerobus::internal Namespace Reference       3.         6.2.1 Detailed Description       3.         6.2.2 Typedef Documentation       3.         6.2.2.1 make_index_sequence_reverse       3.         6.2.2.2 type_at_t       3.         6.2.3 Function Documentation       3.         6.2.3.1 index_sequence_reverse()       3.         6.2.4 Variable Documentation       3.         6.2.4.1 is_instantiation_of_v       3.         6.3 aerobus::known_polynomials Namespace Reference       3.         6.3.1 Detailed Description       3.         6.3.2 Enumeration Type Documentation       3.         7 Concept Documentation       3.         7.1 aerobus::IsEuclideanDomain Concept Reference       3.         7.1.1 Concept definition       3.         7.1.2 Detailed Description       3.
6.2 aerobus::internal Namespace Reference       3.         6.2.1 Detailed Description       3.         6.2.2 Typedef Documentation       3.         6.2.2.1 make_index_sequence_reverse       3.         6.2.2.2 type_at_t       3.         6.2.3 Function Documentation       3.         6.2.3.1 index_sequence_reverse()       3.         6.2.4 Variable Documentation       3.         6.2.4.1 is_instantiation_of_v       3.         6.3 aerobus::known_polynomials Namespace Reference       3.         6.3.1 Detailed Description       3.         6.3.2 Enumeration Type Documentation       3.         7 Concept Documentation       3.         7.1 aerobus::lsEuclideanDomain Concept Reference       3.         7.1.1 Concept definition       3.         7.1.2 Detailed Description       3.
6.2.1 Detailed Description       36         6.2.2 Typedef Documentation       31         6.2.2.1 make_index_sequence_reverse       32         6.2.2.2 type_at_t       33         6.2.3 Function Documentation       36         6.2.3.1 index_sequence_reverse()       36         6.2.4 Variable Documentation       3         6.2.4.1 is_instantiation_of_v       3         6.3 aerobus::known_polynomials Namespace Reference       3         6.3.1 Detailed Description       3         6.3.2 Enumeration Type Documentation       3         7 Concept Documentation       3         7.1 aerobus::IsEuclideanDomain Concept Reference       3         7.1.1 Concept definition       3         7.1.2 Detailed Description       3
6.2.2 Typedef Documentation       36         6.2.2.1 make_index_sequence_reverse       31         6.2.2.2 type_at_t       36         6.2.3 Function Documentation       31         6.2.3.1 index_sequence_reverse()       36         6.2.4 Variable Documentation       31         6.2.4.1 is_instantiation_of_v       33         6.3 aerobus::known_polynomials Namespace Reference       3         6.3.1 Detailed Description       3         6.3.2 Enumeration Type Documentation       3         7 Concept Documentation       3         7.1 aerobus::lsEuclideanDomain Concept Reference       3         7.1.1 Concept definition       3         7.1.2 Detailed Description       3
6.2.2.1 make_index_sequence_reverse       36         6.2.2.2 type_at_t       36         6.2.3 Function Documentation       36         6.2.3.1 index_sequence_reverse()       36         6.2.4 Variable Documentation       36         6.2.4.1 is_instantiation_of_v       36         6.3 aerobus::known_polynomials Namespace Reference       36         6.3.1 Detailed Description       36         6.3.2 Enumeration Type Documentation       36         6.3.2.1 hermite_kind       36         7 Concept Documentation       37         7.1.1 concept definition       36         7.1.2 Detailed Description       36
6.2.2.2 type_at_t       36         6.2.3 Function Documentation       36         6.2.3.1 index_sequence_reverse()       33         6.2.4 Variable Documentation       3         6.2.4.1 is_instantiation_of_v       3         6.3 aerobus::known_polynomials Namespace Reference       3         6.3.1 Detailed Description       3         6.3.2 Enumeration Type Documentation       3         6.3.2.1 hermite_kind       3         7 Concept Documentation       3         7.1.1 concept definition       3         7.1.2 Detailed Description       3
6.2.3 Function Documentation       36         6.2.3.1 index_sequence_reverse()       36         6.2.4 Variable Documentation       37         6.2.4.1 is_instantiation_of_v       37         6.3 aerobus::known_polynomials Namespace Reference       38         6.3.1 Detailed Description       38         6.3.2 Enumeration Type Documentation       38         6.3.2.1 hermite_kind       38         7 Concept Documentation       38         7.1.1 Concept definition       38         7.1.2 Detailed Description       38
6.2.3.1 index_sequence_reverse()       36         6.2.4 Variable Documentation       3         6.2.4.1 is_instantiation_of_v       3         6.3 aerobus::known_polynomials Namespace Reference       3         6.3.1 Detailed Description       3         6.3.2 Enumeration Type Documentation       3         6.3.2.1 hermite_kind       3         7 Concept Documentation       3         7.1 aerobus::IsEuclideanDomain Concept Reference       3         7.1.1 Concept definition       3         7.1.2 Detailed Description       3
6.2.4 Variable Documentation       3         6.2.4.1 is_instantiation_of_v       3         6.3 aerobus::known_polynomials Namespace Reference       3         6.3.1 Detailed Description       3         6.3.2 Enumeration Type Documentation       3         6.3.2.1 hermite_kind       3         7 Concept Documentation       3         7.1 aerobus::IsEuclideanDomain Concept Reference       3         7.1.1 Concept definition       3         7.1.2 Detailed Description       3
6.2.4.1 is_instantiation_of_v       3         6.3 aerobus::known_polynomials Namespace Reference       3         6.3.1 Detailed Description       3         6.3.2 Enumeration Type Documentation       3         6.3.2.1 hermite_kind       3         7 Concept Documentation       3         7.1 aerobus::IsEuclideanDomain Concept Reference       3         7.1.1 Concept definition       3         7.1.2 Detailed Description       3
6.3 aerobus::known_polynomials Namespace Reference       3         6.3.1 Detailed Description       3         6.3.2 Enumeration Type Documentation       3         6.3.2.1 hermite_kind       3         7 Concept Documentation       3         7.1 aerobus::IsEuclideanDomain Concept Reference       3         7.1.1 Concept definition       3         7.1.2 Detailed Description       3
6.3.1 Detailed Description
6.3.2 Enumeration Type Documentation 3 6.3.2.1 hermite_kind 3  7 Concept Documentation 3 7.1 aerobus::IsEuclideanDomain Concept Reference 3 7.1.1 Concept definition 3 7.1.2 Detailed Description 3
6.3.2.1 hermite_kind 3  7 Concept Documentation 3  7.1 aerobus::IsEuclideanDomain Concept Reference 3  7.1.1 Concept definition 3  7.1.2 Detailed Description 3
7 Concept Documentation 7.1 aerobus::IsEuclideanDomain Concept Reference
7.1 aerobus::IsEuclideanDomain Concept Reference
7.1 aerobus::IsEuclideanDomain Concept Reference
7.1.2 Detailed Description
7.2 aerobus::IsField Concept Reference
7.2.1 Concept definition
7.2.2 Detailed Description
7.3 aerobus::IsRing Concept Reference
7.3.1 Concept definition
7.3.2 Detailed Description
8 Class Documentation 4
8.1 aerobus::polynomial < Ring >::val < coeffN >::coeff at < index, E > Struct Template Reference 4
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)> >
8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference
8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference
8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference

8.5 aerobus::ContinuedFraction< a0 > Struct Template Reference	42
8.5.1 Detailed Description	42
8.5.2 Member Typedef Documentation	43
8.5.2.1 type	43
8.5.3 Member Data Documentation	43
8.5.3.1 val	43
8.6 aerobus::ContinuedFraction< a0, rest > Struct Template Reference	43
8.6.1 Detailed Description	43
8.6.2 Member Typedef Documentation	44
8.6.2.1 type	44
8.6.3 Member Data Documentation	44
8.6.3.1 val	44
8.7 aerobus::ConwayPolynomial Struct Reference	44
8.8 aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner< index, ghost > Struct Template Reference	44
8.8.1 Member Function Documentation	44
8.8.1.1 func()	44
8.9 aerobus::polynomial < Ring >::compensated_horner < arithmeticType, P >::EFTHorner <-1, ghost > Struct Template Reference	45
8.9.1 Member Function Documentation	45
8.9.1.1 func()	45
8.10 aerobus::Embed< Small, Large, E > Struct Template Reference	45
8.10.1 Detailed Description	45
8.11 aerobus::Embed< i32, i64 > Struct Reference	45
8.11.1 Detailed Description	46
8.11.2 Member Typedef Documentation	46
8.11.2.1 type	46
8.12 aerobus::Embed< polynomial< Small >, polynomial< Large > > Struct Template Reference	46
8.12.1 Detailed Description	46
8.12.2 Member Typedef Documentation	46
8.12.2.1 type	46
8.13 aerobus::Embed < q32, q64 > Struct Reference	47
8.13.1 Detailed Description	47
8.13.2 Member Typedef Documentation	47
8.13.2.1 type	47
8.14 aerobus::Embed< Quotient< Ring, X >, Ring > Struct Template Reference	47
8.14.1 Detailed Description	48
8.14.2 Member Typedef Documentation	48
8.14.2.1 type	48
8.15 aerobus::Embed< Ring, FractionField< Ring >> Struct Template Reference	48
8.15.1 Detailed Description	48
8.15.2 Member Typedef Documentation	49
8.15.2.1 type	49

8.16 aerobus::Embed $<$ zpz $<$ x $>$ , i32 $>$ Struct Template Reference	49
8.16.1 Detailed Description	49
8.16.2 Member Typedef Documentation	49
8.16.2.1 type	49
8.17 aerobus::polynomial < Ring >::horner_reduction_t < P > Struct Template Reference	50
8.17.1 Detailed Description	50
8.18 aerobus::i32 Struct Reference	50
8.18.1 Detailed Description	51
8.18.2 Member Typedef Documentation	51
8.18.2.1 add_t	51
8.18.2.2 div_t	52
8.18.2.3 eq_t	52
8.18.2.4 gcd_t	52
8.18.2.5 gt_t	52
8.18.2.6 inject_constant_t	52
8.18.2.7 inject_ring_t	53
8.18.2.8 inner_type	53
8.18.2.9 lt_t	53
8.18.2.10 mod_t	53
8.18.2.11 mul_t	53
8.18.2.12 one	53
8.18.2.13 pos_t	53
8.18.2.14 sub_t	54
8.18.2.15 zero	54
8.18.3 Member Data Documentation	54
8.18.3.1 eq_v	54
8.18.3.2 is_euclidean_domain	54
8.18.3.3 is_field	54
8.18.3.4 pos_v	54
8.19 aerobus::i64 Struct Reference	54
8.19.1 Detailed Description	56
8.19.2 Member Typedef Documentation	56
8.19.2.1 add_t	56
8.19.2.2 div_t	56
8.19.2.3 eq_t	56
8.19.2.4 gcd_t	57
8.19.2.5 gt_t	57
8.19.2.6 inject_constant_t	57
8.19.2.7 inject_ring_t	57
8.19.2.8 inner_type	57
8.19.2.9 lt_t	58
8.19.2.10 mod_t	58

8.19.2.11 mul_t	58
8.19.2.12 one	58
8.19.2.13 pos_t	58
8.19.2.14 sub_t	58
8.19.2.15 zero	59
8.19.3 Member Data Documentation	59
8.19.3.1 eq_v	59
8.19.3.2 gt_v	59
8.19.3.3 is_euclidean_domain	59
8.19.3.4 is_field	59
8.19.3.5 lt_v	59
8.19.3.6 pos_v	60
8.20 aerobus::polynomial < Ring >::horner_reduction_t < P >::inner < index, stop > Struct Template Reference	60
8.20.1 Member Typedef Documentation	60
8.20.1.1 type	60
8.21 aerobus::polynomial < Ring >::horner_reduction_t < P >::inner < stop, stop > Struct Template Ref-	
erence	60
8.21.1 Member Typedef Documentation	60
8.21.1.1 type	60
8.22 aerobus::is_prime $<$ n $>$ Struct Template Reference	61
8.22.1 Detailed Description	61
8.22.2 Member Data Documentation	61
8.22.2.1 value	61
8.23 aerobus::polynomial < Ring > Struct Template Reference	61
8.23.1 Detailed Description	63
8.23.2 Member Typedef Documentation	63
8.23.2.1 add_t	63
8.23.2.2 derive_t	63
8.23.2.3 div_t	63
8.23.2.4 eq_t	64
8.23.2.5 gcd_t	64
8.23.2.6 gt_t	64
8.23.2.7 inject_constant_t	64
8.23.2.8 inject_ring_t	64
8.23.2.9 lt_t	65
8.23.2.10 mod_t	65
8.23.2.11 monomial_t	65
8.23.2.12 mul_t	65
8.23.2.13 one	66
8.23.2.14 pos_t	66
8.23.2.15 simplify_t	66
8.23.2.16 sub_t	66

8.23.2.17 X	66
8.23.2.18 zero	66
8.23.3 Member Data Documentation	67
8.23.3.1 is_euclidean_domain	67
8.23.3.2 is_field	67
8.23.3.3 pos_v	67
8.24 aerobus::type_list< Ts >::pop_front Struct Reference	67
8.24.1 Detailed Description	67
8.24.2 Member Typedef Documentation	67
8.24.2.1 tail	67
8.24.2.2 type	67
8.25 aerobus::Quotient $<$ Ring, X $>$ Struct Template Reference	68
8.25.1 Detailed Description	69
8.25.2 Member Typedef Documentation	69
8.25.2.1 add_t	69
8.25.2.2 div_t	69
8.25.2.3 eq_t	69
8.25.2.4 inject_constant_t	70
8.25.2.5 inject_ring_t	70
8.25.2.6 mod_t	70
8.25.2.7 mul_t	70
8.25.2.8 one	70
8.25.2.9 pos_t	71
8.25.2.10 zero	71
8.25.3 Member Data Documentation	71
8.25.3.1 eq_v	71
8.25.3.2 is_euclidean_domain	71
8.25.3.3 pos_v	71
8.26 aerobus::type_list< Ts >::split< index > Struct Template Reference	72
8.26.1 Detailed Description	72
8.26.2 Member Typedef Documentation	72
8.26.2.1 head	72
8.26.2.2 tail	72
8.27 aerobus::type_list< Ts > Struct Template Reference	72
8.27.1 Detailed Description	73
8.27.2 Member Typedef Documentation	73
8.27.2.1 at	73
8.27.2.2 concat	73
8.27.2.3 insert	74
8.27.2.4 push_back	74
8.27.2.5 push_front	74
8.27.2.6 remove	74

8.27.3 Member Data Documentation	74
8.27.3.1 length	74
8.28 aerobus::type_list<> Struct Reference	75
8.28.1 Detailed Description	75
8.28.2 Member Typedef Documentation	75
8.28.2.1 concat	75
8.28.2.2 insert	75
8.28.2.3 push_back	75
8.28.2.4 push_front	75
8.28.3 Member Data Documentation	75
8.28.3.1 length	75
8.29 aerobus::i32::val $<$ x $>$ Struct Template Reference	76
8.29.1 Detailed Description	76
8.29.2 Member Typedef Documentation	76
8.29.2.1 enclosing_type	76
8.29.2.2 is_zero_t	76
8.29.3 Member Function Documentation	76
8.29.3.1 get()	76
8.29.3.2 to_string()	78
8.29.4 Member Data Documentation	78
8.29.4.1 v	78
8.30 aerobus::i64::val < x > Struct Template Reference	78
8.30.1 Detailed Description	78
8.30.2 Member Typedef Documentation	79
8.30.2.1 enclosing_type	79
8.30.2.2 inner_type	79
8.30.2.3 is_zero_t	79
8.30.3 Member Function Documentation	79
8.30.3.1 get()	79
8.30.3.2 to_string()	79
8.30.4 Member Data Documentation	79
8.30.4.1 v	79
8.31 aerobus::polynomial < Ring >::val < coeffN, coeffs > Struct Template Reference	80
8.31.1 Detailed Description	80
8.31.2 Member Typedef Documentation	81
8.31.2.1 aN	81
8.31.2.2 coeff_at_t	81
8.31.2.3 enclosing_type	81
8.31.2.4 is_zero_t	81
8.31.2.5 ring_type	81
8.31.2.6 strip	81
8.31.2.7 value_at_t	81

8.31.3 Member Function Documentation	82
8.31.3.1 compensated_eval()	82
8.31.3.2 eval()	82
8.31.3.3 to_string()	82
8.31.4 Member Data Documentation	83
8.31.4.1 degree	83
8.31.4.2 is_zero_v	83
8.32 aerobus::Quotient < Ring, X >::val < V > Struct Template Reference	83
8.32.1 Detailed Description	83
8.32.2 Member Typedef Documentation	83
8.32.2.1 raw_t	83
8.32.2.2 type	84
8.33 aerobus::zpz::val< x > Struct Template Reference	84
8.33.1 Detailed Description	84
8.33.2 Member Typedef Documentation	84
8.33.2.1 enclosing_type	84
8.33.2.2 is_zero_t	85
8.33.3 Member Function Documentation	85
8.33.3.1 get()	85
8.33.3.2 to_string()	85
8.33.4 Member Data Documentation	85
8.33.4.1 is_zero_v	85
8.33.4.2 v	85
8.34 aerobus::polynomial< Ring >::val< coeffN > Struct Template Reference	85
8.34.1 Detailed Description	86
8.34.2 Member Typedef Documentation	86
8.34.2.1 aN	86
8.34.2.2 coeff_at_t	87
8.34.2.3 enclosing_type	87
8.34.2.4 is_zero_t	87
8.34.2.5 ring_type	87
8.34.2.6 strip	87
8.34.2.7 value_at_t	87
8.34.3 Member Function Documentation	87
8.34.3.1 compensated_eval()	87
8.34.3.2 eval()	87
8.34.3.3 to_string()	88
8.34.4 Member Data Documentation	88
8.34.4.1 degree	88
8.34.4.2 is_zero_v	88
8.35 aerobus::zpz Struct Template Reference	88

Index	199
10.8 examples/compensated_horner.cpp	. 196
10.7 examples/polynomials_over_finite_field.cpp	
10.6 examples/make_polynomial.cpp	
10.5 examples/modular_arithmetic.cpp	
10.4 examples/continued_fractions.cpp	
10.3 examples/fp16.cu	
10.2 examples/custom_taylor.cpp	
10.1 examples/hermite.cpp	
10 Examples	191
9.5 examples.h	. 188
9.4 src/examples.h File Reference	
9.3 aerobus.h	
9.2 src/aerobus.h File Reference	
9.1 README.md File Reference	. 95
9 File Documentation	95
8.35.3.6 pos_v	. 93
8.35.3.5 lt_v	
8.35.3.4 is_field	
8.35.3.3 is_euclidean_domain	
8.35.3.2 gt_v	
8.35.3.1 eq_v	
8.35.3 Member Data Documentation	
8.35.2.14 zero	
8.35.2.13 sub_t	
8.35.2.12 pos_t	
8.35.2.11 one	
8.35.2.10 mul_t	
8.35.2.9 mod_t	. 91
8.35.2.8 lt_t	
8.35.2.7 inner_type	
8.35.2.6 inject_constant_t	. 91
8.35.2.5 gt_t	. 91
8.35.2.4 gcd_t	. 90
8.35.2.3 eq_t	. 90
8.35.2.2 div_t	. 90
8.35.2.1 add_t	. 90
8.35.2 Member Typedef Documentation	. 90

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

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

As we could have using zpz<17>.

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

#### 1.4 Misc

### 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  $\pi$ , e,  $\sqrt{2}$  and  $\sqrt{3}$ : constexpr double A\_SQRT3 = aerobus::SQRT3\_fraction::val; // 1.7320508075688772935

1.5 CUDA 7

### **1.5 CUDA**

When compiled with nvcc and the flag WITH\_CUDA\_FP16, Aerobus provides some 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>.

<><<<< HEAD More, it's (at this time), not easily possible to make it work for \_\_half2 because of another bug.

A workaround is to modify <code>cuda\_fp16.h</code> and add a constexpr modifier to line 5039. This works but only tested on Linux with CUDA 16.1.

Once done, nvcc generates splendid assembly, same as for double or float:

```
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875;
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625;
HFMA2.MMA R5, R6, R5, 0.1666259765625, 0.1666259765625;
HFMA2.MMA R5, R6, R5, 0.5, 0.5;
HFMA2.MMA R5, R6, R5, R7, 1;
HFMA2.MMA R5, R6, R5, RZ;
HFMA2.MMA R5, R6, R5, RZ;
HFMA2.MMA R5, R6, R5, RZ;
HFMA2.MMA R7, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875;
HFMA2.MMA R7, R5, R7, 0.041656494140625, 0.041656494140625;
HFMA2.MMA R7, R5, R7, 0.1666259765625, 0.1666259765625;
HFMA2.MMA R7, R5, R7, 0.5, 0.5;
HFMA2.MMA R7, R5, R7, 1, 1;
HFMA2.MMA R7, R5, R7, R7, 1, 1;
HFMA2.MMA R7, R5, R7, R7, R7, R1, R1, HFMA2.MMA R7, R5, R7, R7, R1, R1, R1, HFMA2.MMA R7, R5, R7, R2.H0_H0;
```

HFMA2 R7, R5, R7, RZ.H0 H0;
More, it's (at this time), not easy to make it work for \_\_half2 because of another bug.

One workaround is to add <code>constexpr</code> modifier on line 5039 of file cuda\_fp16.h. Once done, `examples\fp16.cu" compiles and generates proper assembly. <blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote><blockquote

8 Introduction

# **Namespace Index**

# 2.1 Namespace List

lere is a list of a	all namespaces with brief descriptions:	
aerobus		
Ma	ain namespace for all publicly exposed types or functions	1
aerobus::inte	ernal	
Inte	ernal implementations, subject to breaking changes without notice	3
aerobus::knc	own_polynomials	
Far	milies of well known polynomials such as Hermite or Bernstein	3

10 Namespace Index

# **Concept Index**

# 3.1 Concepts

Here is a list of all concepts with brief descriptions:	
aerobus::IsEuclideanDomain	
Concept to express R is an euclidean domain	39
aerobus::IsField	
Concept to express R is a field	39
aerobus::IsRing	
Concept to express B is a Bing	39

12 Concept Index

# **Class Index**

# 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:	
aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E >	41
aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0  index	> 0)> >
41	
aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> >	41
aerobus::ContinuedFraction < values >	
Continued fraction a0 + $\frac{1}{a_1 + \frac{1}{a_2 + \dots}}$	42
aerobus::ContinuedFraction $<$ a0 $>$	
Specialization for only one coefficient, technically just 'a0'	42
aerobus::ContinuedFraction < a0, rest >	
Specialization for multiple coefficients (strictly more than one)	43
aerobus::ConwayPolynomial	
aerobus::polynomial < Ring >::compensated_horner < arithmeticType, P >::EFTHorner < index, gl	
aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner<-1, ghost	
aerobus::Embed< Small, Large, E >	
Embedding - struct forward declaration	45
aerobus::Embed< i32, i64 >	
Embeds i32 into i64	45
aerobus::Embed< polynomial< Small >, polynomial< Large >>	
Embeds polynomial < Small > into polynomial < Large >	46
aerobus::Embed < q32, q64 >	
Embeds q32 into q64	47
aerobus::Embed< Quotient< Ring, X >, Ring >	
Embeds Quotient <ring, x=""> into Ring</ring,>	47
aerobus::Embed < Ring, FractionField < Ring > >	
Embeds values from Ring to its field of fractions	48
aerobus::Embed< zpz< x >, i32 >	
Embeds zpz values into i32	49
aerobus::polynomial < Ring >::horner_reduction_t < P >	
Used to evaluate polynomials over a value in Ring	50
aerobus::i32	
32 bits signed integers, seen as a algebraic ring with related operations	50
aerobus::i64	
64 bits signed integers, seen as a algebraic ring with related operations	54
aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop >	60
aerobus::polynomial < Ring >::horner_reduction_t < P >::inner < stop, stop >	60
aerobus::is_prime< n >	
Checks if n is prime	61
aerobus::polynomial < Ring >	61
aerobus::type_list< Ts >::pop_front	
Pamayos typos from haad of the list	67

14 Class Index

aerobus::Quotient< Ring, X >	
Quotient ring by the principal ideal generated by 'X' With i32 as Ring and i32::val<2> as X,	
Quotient is Z/2Z	68
aerobus::type_list< Ts >::split< index >	
Splits list at index	72
aerobus::type_list< Ts >	
Empty pure template struct to handle type list	72
aerobus::type_list<>	
Specialization for empty type list	75
aerobus::i32::val< x >	
Values in i32, again represented as types	76
aerobus::i64::val < x >	
Values in i64	78
aerobus::polynomial< Ring >::val< coeffN, coeffs >	
Values (seen as types) in polynomial ring	80
aerobus::Quotient< Ring, X >::val< V >	
Projection values in the quotient ring	83
aerobus::zpz::val< x >	
Values in zpz	84
aerobus::polynomial < Ring >::val < coeffN >	
Specialization for constants	85
aerobus::zpz	
Congruence classes of integers modulo p (32 bits)	88

# File Index

# 5.1 File List

Here is a list of all	files	wit	h br	ief (	desc	cript	tior	ıs:												
src/aerobus.h										 										 98
src/examples.h	١.								 	 		 								 188

16 File Index

# **Namespace Documentation**

# 6.1 aerobus Namespace Reference

main namespace for all publicly exposed types or functions

#### **Namespaces**

· namespace internal

internal implementations, subject to breaking changes without notice

• namespace known\_polynomials

families of well known polynomials such as Hermite or Bernstein

#### Classes

```
    struct ContinuedFraction
```

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

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.

using q32 = FractionField < i32 >

32 bits rationals rationals with 32 bits numerator and denominator

#### **Typedefs**

```
- template<typename T , typename A , typename B >
  using gcd_t = typename internal::gcd< T >::template type< A, B >
     computes the greatest common divisor or A and B
• template<typename... vals>
  using vadd_t = typename internal::vadd< vals... >::type
     adds multiple values (v1 + v2 + ... + vn) vals must have same "enclosing_type" and "enclosing_type" must have an
     add_t binary operator
• template<typename... vals>
  using vmul_t = typename internal::vmul < vals... >::type
     multiplies 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 'IsEuclideanDomain' concept
  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 fpq32 = FractionField< polynomial< q32 >>

     rational fractions with 32 bits rational coefficients rational fractions with rationals coefficients (32 bits numerator and
     denominator)

 using q64 = FractionField < i64 >

     64 bits rationals rationals with 64 bits numerator and denominator

 using pi64 = polynomial < i64 >

     polynomial with 64 bits integers coefficients
using pq64 = polynomial < q64 >
     polynomial with 64 bits rationals coefficients

    using fpq64 = FractionField< polynomial< q64 > >

     polynomial with 64 bits rational coefficients
• template<typename Ring , typename v1 , typename v2 >
  using makefraction_t = typename FractionField< Ring >::template val< v1, v2 >
     helper type: the rational V1/V2 in the field of fractions of Ring
• template<typename v >
  using embed int poly in fractions t = typename Embed< polynomial< typename v::ring type >,
  polynomial < FractionField < typename v::ring type >>>::template type < v >
     embed a polynomial with integers coefficients into rational coefficients polynomials
template<int64_t p, int64_t q>
  using make_q64_t = typename q64::template simplify_t< typename q64::val< i64::inject_constant_t< p >,
  i64::inject_constant_t< q >>>
     helper type: make a fraction from numerator and denominator
• template<int32_t p, int32_t q>
  using make_q32_t = typename q32::template simplify_t< typename q32::val< i32::inject_constant_t< p >,
  i32::inject constant t < q > >
     helper type: make a fraction from numerator and denominator

    template<typename Ring , typename v1 , typename v2 >

  using addfractions_t = typename FractionField< Ring >::template add_t< v1, v2 >
     helper type : adds two fractions

    template<typename Ring , typename v1 , typename v2 >

  using mulfractions_t = typename FractionField< Ring >::template mul_t< v1, v2 >
     helper type: multiplies two fractions
template<typename Ring , auto... xs>
  using make_int_polynomial_t = typename polynomial < Ring >::template val < typename Ring::template
  inject_constant_t< xs >... >
     make a polynomial with coefficients in Ring

    template<typename Ring , auto... xs>

  using make_frac_polynomial_t = typename polynomial < FractionField < Ring > >::template val < typename
  FractionField < Ring >::template inject_constant_t < xs >... >
     make a polynomial with coefficients in FractionField<Ring>
• template<typename T , size_t i>
  using factorial t = typename internal::factorial < T, i >::type
     computes factorial(i), as type
• template<typename T , size_t k, size_t n>
  using combination_t = typename internal::combination< T, k, n >::type
     computes binomial coefficient (k among n) as type
• template<typename T, size t n>
  using bernoulli t = typename internal::bernoulli < T, n >::type
     nth bernoulli number as type in T
template<typename T, size_t n>
  using bell_t = typename internal::bell_helper< T, n >::type
     Bell numbers.
```

```
• template<typename T , int k>
  using alternate_t = typename internal::alternate< T, k >::type
     (-1)^{\wedge}k as type in T
• template<typename T , int n, int k>
  using stirling 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 >
      e^x - 1
• template<typename Integers , size t deg>
  using lnp1 = taylor< Integers, internal::lnp1_coeff, deg >
     ln(1+x)
• template<typename Integers , size_t deg>
  using atan = taylor < Integers, internal::atan_coeff, deg >
     \arctan(x)
• template<typename Integers , size_t deg>
  using sin = taylor< Integers, internal::sin_coeff, deg >
• template<typename Integers , size_t deg>
  using sinh = taylor < Integers, internal::sh coeff, deg >
     sinh(x)
• template<typename Integers , size_t deg>
  using cosh = taylor < Integers, internal::cosh_coeff, deg >
     \cosh(x) hyperbolic cosine
• template<typename Integers , size_t deg>
  using cos = taylor < Integers, internal::cos coeff, deg >
     \cos(x) cosinus
• template<typename Integers , size_t deg>
  using geometric_sum = taylor< Integers, internal::geom_coeff, deg >
      \frac{1}{1-x} zero development of \frac{1}{1-x}
• template<typename Integers , size_t deg>
  using asin = taylor < Integers, internal::asin coeff, deg >
     \arcsin(x) arc sinus
• template<typename Integers , size_t deg>
  using asinh = taylor < Integers, internal::asinh coeff, deg >
     \operatorname{arcsinh}(x) arc hyperbolic sinus
• template<typename Integers , size_t deg>
  using atanh = taylor < Integers, internal::atanh_coeff, deg >
```

#### **Functions**

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

#### **Variables**

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

#### 6.1.1 Detailed Description

main namespace for all publicly exposed types or functions

#### 6.1.2 Typedef Documentation

#### 6.1.2.1 abs\_t

```
template<typename val >
using aerobus::abs_t = typedef std::conditional_t< val::enclosing_type::template pos_v<val>,
val, typename val::enclosing_type::template sub_t<typename val::enclosing_type::zero, val> >
computes absolute value of 'val' val must be a 'value' in a Ring satisfying 'IsEuclideanDomain' concept
```

### **Template Parameters**

```
val a value in a RIng, such as i64::val<-2>
```

#### 6.1.2.2 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

```
\label{template} $$ \text{template}$$ < \text{typename v1 , typename v2 > } $$ using $$ \text{aerobus::addfractions}$$_t = typedef typename $$ FractionField < Ring > ::template add_t < v1, v2 > $$ helper type : adds two fractions $$ $$
```

#### **Template Parameters**

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

## 6.1.2.4 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> \arcsinh(x) arc hyperbolic sinus
```

# **Template Parameters**

Integers	Ring type (for example i64)
----------	-----------------------------

#### **Template Parameters**

#### 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

```
\label{template} $$ \ensuremath{\sf template}$ $$ \ensuremath{\sf template}$ : bernoulli_t = typedef typename internal::bernoulli<T, n>::type $$ $$ \ensuremath{\sf 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) 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
Γ	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> approximation of e
```

#### 6.1.2.16 embed\_int\_poly\_in\_fractions\_t

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::expm1 = typedef typename polynomial<FractionField<Integers> >::template sub_t< exp<Integers, deg>, typename polynomial<FractionField<Integers> >::one> e^x-1
```

#### **Template Parameters**

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

# 6.1.2.19 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**

T	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 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} 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

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
```

X	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 \pi 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

#### 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> 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.

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

# 6.1.2.45 stirling\_2\_t

```
template<typename T , int n, int k>
using aerobus::stirling_2_t = typedef typename internal::stirling_2_helper<T, n, k>::type
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**

X	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
```

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	

#### 6.1.2.50 vadd\_t

#### **Template Parameters**

vala	
vais	

#### 6.1.2.51 vmul\_t

#### **Template Parameters**

```
...vals
```

#### 6.1.3 Function Documentation

#### 6.1.3.1 aligned\_malloc()

'portable' aligned allocation of count elements of type T

# **Template Parameters**

T	the type of elements to store
	the type of elements to eleme

#### **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 or value from T
```

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

#### **Template Parameters**

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

#### 6.1.4.2 bernoulli\_v

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

#### nth bernoulli number as value in FloatType

#### **Template Parameters**

FloatType	(double or float for example)
T	(aerobus::i64 for example)
n	

#### 6.1.4.3 combination\_v

```
template<typename T , size_t k, size_t n>
constexpr T::inner_type aerobus::combination_v = internal::combination<T, k, n>::value [inline],
[constexpr]
```

computes binomial coefficients (k among n) as value

#### **Template Parameters**

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

T (aerobus::i64 for example)
------------------------------

i	
---	--

# 6.2 aerobus::internal Namespace Reference

internal implementations, subject to breaking changes without notice

#### **Classes**

```
    struct FractionField

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

· struct is prime
struct _is_prime< 0, i >

    struct _is_prime< 1, i >

    struct _is_prime< 2, i >

    struct _is_prime< 3, i >

    struct _is_prime< 5, i >

    struct _is_prime< 7, i >

• struct _is_prime< n, i, std::enable_if_t<(n !=2 &&n !=3 &&n % 2 !=0 &&n % 3==0)>>
• struct is prime < n, i, std::enable if t < (n != 2 \&\&n \% 2==0) > >

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

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

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

· struct atan coeff

    struct atan coeff helper

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

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

· struct atanh_coeff
· struct atanh coeff helper

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

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

    struct bell_helper

    struct bell_helper< T, 0 >

    struct bell helper< T, 1 >
```

• struct bell helper < T, n, std::enable if t < (n > 1) > >

· struct bernoulli struct bernoulli < T, 0 > · struct bernoulli coeff · struct bernoulli\_helper struct bernoulli helper< T, accum, m, m > struct bernstein\_helper struct bernstein\_helper< 0, 0, I > • struct bernstein helper< i, m, I, std::enable if t<(m > 0) &&(i > 0) &&(i < m)> > struct bernstein helper< i, m, l, std::enable if t<(m > 0) &&(i==0)>> struct bernstein\_helper< i, m, I, std::enable\_if\_t<(m > 0) &&(i==m)> > struct 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 FloatLayout < long double > 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 fma\_helper< long double > 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, I > struct hermite\_helper< deg, known\_polynomials::hermite\_kind::probabilist, l > struct insert\_h • struct is\_instantiation\_of struct is\_instantiation\_of< TT, TT< Ts... >> • struct laguerre helper struct laguerre\_helper< 0, I > struct laguerre\_helper< 1, I > struct legendre helper struct legendre\_helper< 0, I > struct legendre helper< 1, I > struct Inp1\_coeff struct Inp1\_coeff< T, 0 > struct make taylor impl • struct make\_taylor\_impl< T, coeff\_at, std::integer\_sequence< size\_t, ls... >> struct pop\_front\_h · struct pow • struct pow< T, p, n, std::enable if t < n==0 > > struct pow< T, p, n, std::enable\_if\_t<(n % 2==1)>> struct pow< T, p, n, std::enable\_if\_t<(n > 0 &&n % 2==0)> > struct pow\_scalar · struct remove h struct sh coeff · struct sh coeff helper struct sh\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==0 >> struct sh\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==1 >> struct sin coeff • struct sin\_coeff\_helper struct sin\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==0 >> struct sin\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==1 >> · struct Split 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 >{}))

#### **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

# 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

```
template<typename R>
concept aerobus::IsField = IsEuclideanDomain<R> && requires {
    R::is_field == true;
}
```

# 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'.

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)

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 DEVICE void func (arithmeticType x, arithmeticType \*pi, arithmeticType \*sigma, arithmetic

Type \*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

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> representation 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_\hookleftarrow 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

```
\begin{array}{l} \text{embeds Quotient} {<} \text{Ring, X} {>} \text{ into Ring} \\ \text{\#include } {<} \text{aerobus.h} {>} \end{array}
```

#### **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

```
\label{template} \begin{tabular}{ll} template < typename Ring > \\ struct aerobus:: Embed < Ring, FractionField < Ring > > \\ \end{tabular}
```

**Template Parameters** 

```
Ring an integers ring, such as i32
```

embeds values from Ring to its field of fractions

# 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

```
\label{template} $$ \end{template} < int32_t \ x>$ struct aerobus::Embed < zpz < x>, i32>$ embeds zpz values into i32
```

#### **Template Parameters**

```
x an integer
```

# 8.16.2 Member Typedef Documentation

#### 8.16.2.1 type

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

#### **Template Parameters**

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

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

· src/aerobus.h

# 8.17 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**

```
• using inner_type = int32_t
• using zero = val< 0 >
     constant zero
• using one = val< 1 >
     constant one

    template<auto x>

  using inject_constant_t = val < static_cast < int32_t >(x)>
     inject a native constant

    template<typename v >

  using inject_ring_t = v

    template<typename v1 , typename v2 >

  using add_t = typename add< v1, v2 >::type
     addition operator yields v1 + v2
• 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 It t = typename It < 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

Generated by Doxygen

#### 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<br/>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
```



#### 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 i32
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

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

static constexpr bool eq\_v = eq\_t <v1, v2>::value

equality operator yields v1 == v2 as boolean value

template<typename v >
 static constexpr bool pos\_v = pos\_t<v>::value
 positivity yields v > 0 as boolean value

# 8.19.1 Detailed Description

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
```

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 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**

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

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 lt\_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
```

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

#### 8.19.3.6 pos\_v

#### **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_
        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_\(\chi\)
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 lt_t = typename lt_helper< v1, v2 >::type
     strict less operator

    template<typename v1 , typename v2 >

  using gt_t = typename gt_helper< v1, v2 >::type
     strict greater operator
• template<typename v1 , typename v2 >
  using div_t = typename div < v1, v2 >::q_type
     division operator

    template<typename v1 , typename v2 >

  using mod_t = typename div_helper< v1, v2, zero, v1 >::mod_type
     modulo operator

    template<typename coeff , size_t deg>

  using monomial t = typename monomial < coeff, deg >::type
     monomial : coeff X^{\wedge} deg
• template<typename v >
  using derive_t = typename derive_helper< v >::type
     derivation operator

    template<typename v >

  using pos_t = typename Ring::template pos_t < typename v::aN >
     checks for positivity (an > 0)
• template<typename v1 , typename v2 >
  using gcd t = std::conditional t < Ring::is euclidean domain, typename make unit < gcd t < polynomial <
  Ring >, v1, v2 > ::type, void >
     greatest common divisor of two polynomials

    template<auto x>

  using inject_constant_t = val< typename Ring::template inject_constant_t < x > >
     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

Χ	

## 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**

V
---

## 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

```
template<typename Ring >
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring >::mod_t = typename div_helper<v1, v2, zero, v1>::mod_type
modulo operator
```

## **Template Parameters**

v1	
v2	

## 8.23.2.11 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

```
\label{template} $$ \end{template} $$ \end{template} $$ \end{template} $$ \end{template} $$ v > $$ \end{template} $$$ \end{template} $$$ \end{template} $$
```

## **Template Parameters**

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 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
        quotien rings are euclidean domain
```

## 8.25.1 Detailed Description

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

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

#### **Template Parameters**

Ring	A ring type, such as 'i32', must satisfy the IsRing concept
X	a value in Ring, such as i32::val<2>

## 8.25.2 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←
::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
::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 \( \text{::type, typename v2::type} > \)
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

index
```

## 8.26.2 Member Typedef Documentation

#### 8.26.2.1 head

```
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**

index	

#### 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**

Ū	

## 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

```
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

index		index	
-------	--	-------	--

## 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<iint32_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<iint32_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()

```
\label{template} $$ \text{template}$ \le int32_t x > $$ \text{static std}::string aerobus::i32::val< x > ::to_string () [inline], [static] $$ \textbf{string representation of value} $$
```

## 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()

```
\label{template} $$ \text{template}_{\text{int64}\_t \ x>} $$ \text{static std}_{\text{::string aerobus}_{\text{::i64}::val}<\ x>::to_{\text{string ()}} $$ [inline], [static] $$ \text{string representation} $$
```

## 8.30.4 Member Data Documentation

## 8.30.4.1 v

```
template<int64_t x>
constexpr int64_t aerobus::i64::val< x >::v = x [static], [constexpr]
actual value
```

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

• src/aerobus.h

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

```
values (seen as types) in polynomial ring
#include <aerobus.h>
```

## **Public Types**

• using ring type = Ring

ring coefficients live in

using enclosing\_type = polynomial < Ring >

enclosing ring type

using aN = coeffN

heavy weight coefficient (non zero)

• using strip = val< coeffs... >

remove largest coefficient

using is\_zero\_t = std::bool\_constant<(degree==0) &&(aN::is\_zero\_t::value)>

true type if polynomial is constant zero

template < size\_t index >

using coeff\_at\_t = typename coeff\_at< index >::type

type of coefficient at index

• template<typename x >

using value\_at\_t = horner\_reduction\_t < val > ::template inner < 0, degree+1 > ::template type < typename Ring::zero, x >

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

static std::string to\_string ()

get a string representation of polynomial

template<typename arithmeticType >

static constexpr DEVICE INLINED arithmeticType eval (const arithmeticType &x)

evaluates polynomial seen as a function operating on arithmeticType

template<typename arithmeticType >

static DEVICE INLINED arithmeticType compensated\_eval (const arithmeticType &x)

Evaluate polynomial on x using compensated horner scheme.

## **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**

mple
1

#### **Parameters**



## 8.31.3.2 eval()

evaluates polynomial seen as a function operating on arithmeticType

#### **Template Parameters**

arithmeticType	usually float or double
----------------	-------------------------

## **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^{\wedge} 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

projection values in the quotient ring

```
\label{eq:top-condition} \begin{split} & template < typename \ Ring, \ typename \ X > \\ & template < typename \ V > \\ & struct \ aerobus:: Quotient < Ring, \ X > :: val < V > \end{split}
```

**Template Parameters** 

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

## 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
```

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]
```

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
```

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

template<typename x >
 using value at t = coeffN

- 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)

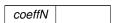
## **Static Public Attributes**

```
    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** 



## 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
```

#### 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_\top 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()

```
template<typename Ring >
template<typename coeffN >
template<typename arithmeticType >
```

#### 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

#### **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)

• template<typename v1 , typename v2 > static constexpr bool lt\_v = lt\_t<v1, v2>::value

strictly smaller operator (booleanvalue) • template<typename v1 , typename v2 >

static constexpr bool eq\_v = eq\_t<v1, v2>::value equality operator (booleanvalue)

• template<typename v > static constexpr bool pos\_v = pos\_t < v > ::value positivity operator (boolean value)

## 8.35.1 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>
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<iint32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz::gcd_t = gcd_t<i32, v1, v2>
greatest common divisor
```

## **Template Parameters**

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

## 8.35.2.5 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)
```

## **Template Parameters**

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<iint32_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
```

## **Template Parameters**

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

## 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
```

## **Template Parameters**

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
```

## 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
```

96 File Documentation

```
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
00056
               return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
00057
               #endif
00058
00059 } // namespace aerobus
00060
00061 // concepts
00062 namespace aerobus {
00064
          template <typename R>
          concept IsRing = requires {
00065
00066
              typename R::one;
00067
               typename R::zero;
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>;
00077
               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>;
00078
00079
08000
               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>;
00084
               R::is_euclidean_domain == true;
00085
          };
00086
          template<typename R>
00089
          concept IsField = IsEuclideanDomain<R> && requires {
          R::is_field == true;
};
00090
00091
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 {
00097
          namespace internal {
               static consteval DEVICE uint16_t my_internal_float2half(
00098
00099
                   const float f, uint32_t &sign, uint32_t &remainder) {
00100
                   uint32_t x;
00101
                   uint32_t u;
00102
                   uint32_t result;
00103
                   x = std::bit_cast<int32_t>(f);
                   u = (x \& 0x7fffffffU);
00104
00105
                   sign = ((x \gg 16U) \& 0x8000U);
                   // NaN/+Inf/-Inf
00106
                   if (u >= 0x7f800000U) {
00108
                       remainder = 0U;
                   result = ((u == 0x7f800000U) ? (sign | 0x7c00U) : 0x7fffU);
} else if (u > 0x477fefffU) { // Overflows
remainder = 0x8000000U;
00109
00110
00111
00112
                       result = (sign | 0x7bffU);
00113
                   } else if (u >= 0x38800000U) { // Normal numbers
                      remainder = u « 19U;
00114
00115
                       u -= 0x38000000U;
00116
                       result = (sign | (u \gg 13U));
                   } else if (u < 0x33000001U) { // +0/-0
00117
00118
                       remainder = u:
00119
                       result = sign;
                   } else { // Denormal numbers
00120
00121
                      const uint32_t exponent = u » 23U;
00122
                       const uint32_t shift = 0x7eU - exponent;
                       uint32_t mantissa = (u & 0x7fffffU);
00123
                       mantissa |= 0x800000U;
00124
                       remainder = mantissa « (32U - shift);
00125
00126
                       result = (sign | (mantissa » shift));
00127
                       result &= 0x0000FFFFU;
00128
00129
                   return static_cast<uint16_t>(result);
00130
              }
00131
00132
               static consteval DEVICE __half my_float2half_rn(const float a) {
                  __half val;
00133
00134
                    _half_raw r;
00135
                   uint32_t sign = 0U;
                  uint32_t remainder = 0U;
r.x = my_internal_float2half(a, sign, remainder);
00136
00137
```

```
if ((remainder > 0x800000000) || ((remainder == 0x800000000) && ((r.x & 0x10) != 00))) {
00139
                      r.x++;
00140
                  }
00141
                  val = std::bit_cast<__half>(r);
00142
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>
              struct int16_convert_helper;
00151
00152
00153
              template <typename Out, int16_t x>
00154
              struct int16_convert_helper<0ut, x,
                  std::enable_if_t<!std::is_same_v<Out, __half> && !std::is_same_v<Out, __half2>> {
00155
00156
                  static constexpr Out value() {
00157
                      return static_cast<Out>(x);
00158
00159
              };
00160
00161
              template <int16 t x>
00162
              struct int16_convert_helper<__half, x> {
                static constexpr __half value()
00163
00164
                      return convert_int16_to_half<x>;
00165
00166
              } ;
00167
00168
              template <int16 t x>
              struct int16_convert_helper<__half2, x> {
   static constexpr __half2 value() {
00169
00170
00171
                      return __half2(convert_int16_to_half<x>, convert_int16_to_half<x>);
00172
              };
00173
00174
00175
          } // namespace internal
00176 } // namespace aerobus
00177 #endif
00178
00179 // cast
00180 namespace aerobus {
00181
         namespace internal {
            template<typename Out, typename In>
00182
00183
              struct staticcast {
                template<auto x>
00184
                  static consteval INLINED DEVICE Out func() {
00185
00186
                      return static_cast<Out>(x);
00187
00188
              };
00189
00190
              #ifdef WITH_CUDA_FP16
00191
              template<>
              struct staticcast<__half, int16_t> {
00192
00193
                 template<int16_t x>
static consteval INLINED DEVICE __half func() {
00194
00195
                      return int16_convert_helper<__half, x>::value();
00196
00197
              };
00198
00199
             template<>
00200
              struct staticcast<__half2, int16_t> {
00201
               template<int16_t x>
static consteval INLINED DEVICE __half2 func() {
00202
00203
                      return int16_convert_helper<__half2, x>::value();
00204
                  }
00205
              };
00206
              #endif
             // namespace internal
00208 } // namespace aerobus
00209
00210 // fma_helper, required because nvidia fails to reconstruct fma for fp16 types
00211 namespace aerobus {
00212
         namespace internal {
00213
             template<typename T>
00214
              struct fma_helper;
00215
00216
              template<>
              struct fma_helper<double> {
00217
                 static constexpr INLINED DEVICE double eval(const double x, const double y, const double
00218
     z) {
00219
                      return x * y + z;
00220
00221
              } ;
00222
00223
              template<>
```

```
struct fma_helper<long double> {
00225
                 static constexpr INLINED DEVICE long double eval(
00226
                      const long double x, const long double y, const long double z) {
00227
                          return x * y + z;
00228
00229
              };
00230
00231
              template<>
00232
              struct fma_helper<float> {
                  static constexpr INLINED DEVICE float eval(const float x, const float y, const float z) {
00233
00234
                       return x * y + z;
00235
00236
              } ;
00237
00238
              template<>
              struct fma_helper<int32_t> {
    static constexpr INLINED DEVICE int16_t eval(const int16_t x, const int16_t y, const
00239
00240
     int16 t z) {
00241
                       return x * y + z;
00242
                  }
00243
              };
00244
00245
              template<>
              struct fma_helper<int16_t> {
00246
                  static constexpr INLINED DEVICE int32_t eval(const int32_t x, const int32_t y, const
00247
     int32_t z) {
00248
                       return x * y + z;
00249
00250
              } ;
00251
00252
              template<>
00253
              struct fma_helper<int64_t> {
                  static constexpr INLINED DEVICE int64_t eval(const int64_t x, const int64_t y, const
00254
     int64_t z) {
00255
                       return x * y + z;
00256
                  }
00257
              };
00259
              #ifdef WITH_CUDA_FP16
00260
              template<>
00261
              struct fma_helper<__half> {
                  static constexpr INLINED DEVICE __half eval(const __half x, const __half y, const __half
00262
z) {
00263
                      #ifdef __CUDA_ARCH_
00264
                       return __hfma(x, y, z);
00265
                      #else
00266
                       return x * y + z;
00267
                       #endif
                 }
00268
00269
              };
00270
              template<>
00271
              struct fma_helper<__half2> {
__half2 z) {
00272
                  static constexpr INLINED DEVICE __half2 eval(const __half2 x, const __half2 y, const
                       #ifdef ___CUDA_ARCH_
00274
                      <u>return</u> <u>hfma2(x, y, z);</u>
00275
                      #else
00276
                       return x * y + z;
00277
                       #endif
              } ;
**
00278
00279
00280
              #endif
00281
             // namespace internal
00282 } // namespace aerobus
00283
00284 // compensated horner utilities
00285 namespace aerobus {
        namespace internal {
00286
00287
             template <typename T>
00288
              struct FloatLayout;
00289
00290
              #ifdef _MSC_VER
00291
              template <>
              struct FloatLayout<long double> {
00292
                  static constexpr uint8_t exponent = 11;
static constexpr uint8_t mantissa = 53;
00293
00294
00295
                  static constexpr uint8_t r = 27; // ceil(mantissa/2)
00296
00297
              #else
00298
              template <>
              struct FloatLayout<long double> {
00299
00300
                  static constexpr uint8_t exponent = 15;
00301
                  static constexpr uint8_t mantissa = 63;
00302
                  static constexpr uint8_t r = 32; // ceil(mantissa/2)
                  static constexpr long double shift = (1LL « r) + 1;
00303
00304
              }:
00305
              #endif
```

```
00306
00307
                template <>
00308
                struct FloatLayout<double> {
00309
                   static constexpr uint8_t exponent = 11;
                    static constexpr uint8_t mantissa = 53;
static constexpr uint8_t r = 27; // ceil(mantissa/2)
00310
00311
                    static constexpr double shift = (1LL « r) + 1;
00312
00313
00314
00315
                template <>
                struct FloatLayout<float> {
00316
00317
                   static constexpr uint8_t exponent = 8;
                     static constexpr uint8_t mantissa = 24;
static constexpr uint8_t r = 11; // ceil(mantissa/2)
00318
00319
00320
                     static constexpr float shift = (1 « r) + 1;
00321
00322
00323
                template<typename T>
                struct Split {
                    static constexpr INLINED DEVICE void func(T a, T *x, T *y) {
00325
                        T z = a * FloatLayout<T>::shift;
*x = z - (z - a);
*y = a - *x;
00326
00327
00328
00329
                     }
00330
                };
00331
00332
                #ifdef WITH_CUDA_FP16
00333
                template<>
00334
                struct Split<__half> {
                    static constexpr INLINED DEVICE void func(_half a, _half *x, _half *y) {
    _half z = a * _half_raw(0x5280); // TODO(JeWaVe): check this value
00335
00336
00337
                          *x = z - (z - a);
00338
00339
00340
                } ;
00341
00342
                template<>
                struct Split<__half2> {
                static constexpr INLINED DEVICE void func(_half2 a, _half2 *x, _half2 *y) {
    __half2 z = a * __half2(_half_raw(0x5280), __half_raw(0x5280)); // TODO(JeWaVe):
00344
check this value
                        *x = z - (z - a);
                         *y = a - *x;
00347
                };
00348
00349
00350
                #endif
00351
00352
                template<typename T>
                static constexpr INLINED DEVICE void two_sum(T a, T b, T *x, T *y) {
00353
00354
                   *x = a + b;
                     Tz = *x - a;
00356
                    *y = (a - (*x - z)) + (b - z);
00357
00358
00359
                template<typename T>
00360
                static constexpr INLINED DEVICE void two_prod(T a, T b, T *x, T *y) {
                   *x = a * b;
#ifdef __clang_
00362
00363
                     *y = fma_helper<T>::eval(a, b, -*x);
00364
                     #else
00365
                     T ah. al. bh. bl:
                     Split<T>::func(a, &ah, &al);
Split<T>::func(b, &bh, &bl);
00366
00367
00368
                     *y = al * bl - (((*x - ah * bh) - al * bh) - ah * bl);
00369
                     #endif
00370
                }
00371
00372
                template<tvpename T, size t N>
                static INLINED DEVICE T horner(T *p1, T *p2, T x) {
00373
                    T r = p1[0] + p2[0];

for (int64_t i = N - 1; i >= 0; --i) {
00374
00375
00376
                         r = r * x + p1[N - i] + p2[N - i];
00377
                    }
00378
00379
                     return r;
00380
           } // namespace internal
00381
00382 } // namespace aerobus
00383
00384 // utilities
00385 namespace aerobus {
           namespace internal {
00387
               template<template<typename...> typename TT, typename T>
00388
                struct is_instantiation_of : std::false_type { };
00389
                template<template<typename...> typename TT, typename... Ts>
struct is_instantiation_of<TT, TT<Ts...» : std::true_type { };</pre>
00390
00391
```

```
00392
               template<template<typename...> typename TT, typename T>
00393
00394
              inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00395
00396
              template <int64_t i, typename T, typename... Ts>
00397
              struct type_at {
                  static_assert(i < sizeof...(Ts) + 1, "index out of range");
using type = typename type_at<i - 1, Ts...>::type;
00398
00399
00400
00401
00402
              template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00403
                  using type = T;
00404
00405
00406
               template <size_t i, typename... Ts>
00407
              using type_at_t = typename type_at<i, Ts...>::type;
00408
00409
00410
              template<size_t n, size_t i, typename E = void>
00411
              struct _is_prime {};
00412
00413
              template<size_t i>
00414
              struct _is_prime<0, i> {
                  static constexpr bool value = false;
00415
00416
00417
00418
              template<size_t i>
00419
              struct _is_prime<1, i> {
00420
                  static constexpr bool value = false;
00421
00422
00423
              template<size_t i>
00424
              struct _is_prime<2, i> {
00425
                  static constexpr bool value = true;
00426
00427
00428
              template<size t i>
              struct _is_prime<3, i> {
00430
                  static constexpr bool value = true;
00431
00432
00433
              template<size t i>
              struct _is_prime<5, i> \{
00434
                  static constexpr bool value = true;
00435
00436
00437
00438
              template<size_t i>
              struct _is_prime<7, i> {
00439
                  static constexpr bool value = true;
00440
00441
00442
00443
              template<size_t n, size_t i>
00444
              struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)» {
00445
                 static constexpr bool value = false;
00446
00447
00448
              template<size_t n, size_t i>
00449
              struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)» {
00450
                  static constexpr bool value = false;
00451
00452
              template<size_t n, size_t i> struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)> {
00453
00454
00455
                  static constexpr bool value = true;
00456
00457
00458
              template<size_t n, size_t i>
              struct _is_prime<n, i, std::enable_if_t<(
    n % i == 0 &&</pre>
00459
00460
                  n >= 9 &&
00461
00462
                   n % 3 != 0 &&
00463
                  n % 2 != 0 &&
00464
                  i * i > n)» {
                  static constexpr bool value = true;
00465
00466
              };
00467
00468
               template<size_t n, size_t i>
00469
               struct _is_prime<n, i, std::enable_if_t<(
00470
                  n % (i+2) == 0 &&
00471
                  n >= 9 &&
00472
                  n % 3 != 0 &&
00473
                  n % 2 != 0 &&
00474
                   i * i <= n) » {
00475
                   static constexpr bool value = true;
00476
              };
00477
00478
              template<size t n, size t i>
```

```
struct _is_prime<n, i, std::enable_if_t<(</pre>
00480
                       n % (i+2) != 0 &&
00481
                       n % i != 0 &&
00482
                       n >= 9 &&
00483
                       n % 3 != 0 &&
00484
                       n % 2 != 0 &&
00485
                       (i * i <= n))» {
00486
                   static constexpr bool value = _is_prime<n, i+6>::value;
00487
00488
          } // namespace internal
00489
00492
          template<size_t n>
00493
          struct is prime {
00495
              static constexpr bool value = internal::_is_prime<n, 5>::value;
00496
00497
00501
          template<size t n>
00502
          static constexpr bool is_prime_v = is_prime<n>::value;
00503
00504
          // gcd
00505
          namespace internal {
00506
              template <std::size_t... Is>
00507
              constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
00508
                   -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
00509
00510
              template <std::size_t N>
              using make_index_sequence_reverse
00511
00512
                  = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00513
00519
              template<typename Ring, typename E = void>
00520
              struct acd:
00521
00522
              template<typename Ring>
00523
              struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
00524
                  template<typename A, typename B, typename E = void>
00525
                   struct gcd_helper {};
00526
                  // B = 0, A > 0
00528
                   template<typename A, typename B>
00529
                   struct gcd_helper<A, B, std::enable_if_t<
00530
                       ((B::is_zero_t::value) &&
00531
                           (Ring::template gt_t<A, typename Ring::zero>::value))» {
00532
                       using type = A;
00533
                   };
00534
00535
                   // B = 0, A < 0
00536
                   template<typename A, typename B>
00537
                   struct gcd_helper<A, B, std::enable_if_t<
00538
                       ((B::is_zero_t::value) &&
                           !(Ring::template gt_t<A, typename Ring::zero>::value))» {
00539
00540
                       using type = typename Ring::template sub_t<typename Ring::zero, A>;
00541
00542
00543
                   // B != 0
                  template<typename A, typename B>
struct gcd_helper<A, B, std::enable_if_t<</pre>
00544
00545
00546
                       (!B::is_zero_t::value)
00547
00548
                   private: // NOLINT
00549
                       // A / B
                       using k = typename Ring::template div_t<A, B>; 
// A - (A/B)*B = A % B
00550
00551
00552
                       using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
00553
00554
                   public:
00555
                      using type = typename gcd_helper<B, m>::type;
00556
00557
                   template<typename A, typename B>
00558
                  using type = typename gcd_helper<A, B>::type;
00560
00561
          } // namespace internal
00562
          // vadd and vmul
00563
00564
          namespace internal {
00565
              template<typename... vals>
00566
              struct vmul {};
00567
00568
              template<typename v1, typename... vals>  
              struct vmul<v1, vals...> {
    using type = typename v1::enclosing_type::template mul_t<v1, typename</pre>
00569
00570
     vmul<vals...>::type>;
00571
             };
00572
00573
              template<typename v1>
00574
              struct vmul<v1> {
00575
                  using type = v1;
```

```
00576
              };
00577
00578
              template<typename... vals>
00579
              struct vadd {};
00580
00581
              template<typename v1, typename... vals>
              struct vadd<v1, vals...> {
00583
                 using type = typename v1::enclosing_type::template add_t<v1, typename
     vadd<vals...>::type>;
00584
             };
00585
00586
              template<tvpename v1>
00587
              struct vadd<v1> {
00588
                using type = v1;
00589
00590
          } // namespace internal
00591
          template<typename T, typename A, typename B>
using gcd_t = typename internal::gcd<T>::template type<A, B>;
00594
00595
00596
00600
          template<typename... vals>
00601
          using vadd_t = typename internal::vadd<vals...>::type;
00602
00606
          template<typename... vals>
00607
          using vmul_t = typename internal::vmul<vals...>::type;
00608
00612
          template<typename val>
00613
          requires IsEuclideanDomain<typename val::enclosing_type>
00614
          using abs_t = std::conditional_t<
00615
                          val::enclosing_type::template pos_v<val>,
                          val, typename val::enclosing_type::template
00616
      sub_t<typename val::enclosing_type::zero, val>>;
00617 } // namespace aerobus
00618
00619 // embedding
00620 namespace aerobus {
          template<typename Small, typename Large, typename E = void>
00625
          struct Embed;
00627 } // namespace aerobus
00628
00629 namespace aerobus {
00634
          template<typename Ring, typename X>
          requires IsRing<Ring>
00635
00636
          struct Quotient {
             template <typename V>
00639
00640
              struct val {
              public:
00641
00642
                  using raw_t = V;
                  using type = abs_t<typename Ring::template mod_t<V, X>>;
00643
00644
00645
00647
              using zero = val<typename Ring::zero>;
00648
00650
              using one = val<typename Ring::one>;
00651
00655
              template<typename v1, typename v2>
              using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00656
00657
00661
              template<typename v1, typename v2>
00662
                          = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00663
00667
              template<typename v1, typename v2>
00668
              using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00669
00673
              template<typename v1, typename v2>
00674
              using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00675
00679
              template<typename v1, typename v2>
00680
              using eg t = typename Ring::template eg t<typename v1::type, typename v2::type>;
00685
              template<typename v1, typename v2>
00686
              static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00687
00691
              template<typename v1>
00692
              using pos_t = std::true_type;
00693
00697
              template<typename v>
00698
              static constexpr bool pos_v = pos_t<v>::value;
00699
00701
              static constexpr bool is euclidean domain = true:
00702
00706
              template<auto x>
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00707
00708
00712
              template<typename v>
00713
              using inject_ring_t = val<v>;
00714
          };
```

```
00715
00719
           template<typename Ring, typename X>
00720
           struct Embed<Quotient<Ring, X>, Ring> {
00723
               template<typename val>
00724
               using type = typename val::raw_t;
00725
00726 }
         // namespace aerobus
00727
00728 // type_list
00729 namespace aerobus {
00731
          template <typename... Ts>
00732
           struct type_list;
00733
00734
           namespace internal {
00735
                template <typename T, typename... Us>
00736
                struct pop_front_h {
                    using tail = type_list<Us...>;
using head = T;
00737
00738
00740
00741
               template <size_t index, typename L1, typename L2>
00742
                struct split_h {
00743
                private:
                    static_assert(index <= L2::length, "index ouf of bounds");</pre>
00744
                    using a = typename L2::pop_front::type;
using b = typename L2::pop_front::tail;
00745
00746
00747
                    using c = typename L1::template push_back<a>;
00748
                public:
00749
                    using head = typename split_h<index - 1, c, b>::head;
using tail = typename split_h<index - 1, c, b>::tail;
00750
00751
00752
               };
00753
00754
                template <typename L1, typename L2>
               struct split_h<0, L1, L2> {
    using head = L1;
00755
00756
00757
                    using tail = L2;
00758
00759
00760
                template <size_t index, typename L, typename T>
00761
                struct insert_h {
                    static_assert(index <= L::length, "index ouf of bounds");</pre>
00762
00763
                    using s = typename L::template split<index>;
00764
                    using left = typename s::head;
00765
                    using right = typename s::tail;
00766
                    using 11 = typename left::template push_back<T>;
00767
                    using type = typename ll::template concat<right>;
00768
               };
00769
00770
               template <size t index, typename L>
00771
               struct remove_h {
00772
                    using s = typename L::template split<index>;
                    using left = typename s::head;
using right = typename s::tail;
00773
00774
00775
                    using rr = typename right::pop_front::tail;
using type = typename left::template concat<rr>;
00776
00777
               } ;
00778
           } // namespace internal
00779
00782
           template <typename... Ts>
00783
           struct type_list {
00784
            private:
00785
               template <typename T>
00786
               struct concat_h;
00787
00788
                template <typename... Us>
               struct concat_h<type_list<Us...» {</pre>
00789
00790
                    using type = type_list<Ts..., Us...>;
00791
00792
00793
            public:
00795
               static constexpr size_t length = sizeof...(Ts);
00796
00799
               template <typename T>
00800
               using push_front = type_list<T, Ts...>;
00801
00804
                template <size_t index>
00805
               using at = internal::type_at_t<index, Ts...>;
00806
00808
                struct pop front {
00810
                   using type = typename internal::pop_front_h<Ts...>::head;
00812
                    using tail = typename internal::pop_front_h<Ts...>::tail;
00813
00814
00817
               template <typename T>
               using push_back = type_list<Ts..., T>;
00818
00819
```

```
template <typename U>
00823
              using concat = typename concat_h<U>::type;
00824
00827
              template <size_t index>
              struct split {
00828
00829
               private:
                  using inner = internal::split_h<index, type_list<>, type_list<Ts...»;
00830
00831
00832
               public:
00833
                  using head = typename inner::head;
                  using tail = typename inner::tail;
00834
00835
00836
00840
              template <typename T, size_t index>
00841
              using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00842
              template <size_t index>
00845
00846
              using remove = typename internal::remove_h<index, type_list<Ts...»::type;</pre>
00847
          };
00848
00850
          template <>
00851
          struct type_list<> {
00852
              static constexpr size_t length = 0;
00853
00854
              template <typename T>
00855
              using push_front = type_list<T>;
00856
00857
              template <typename T>
00858
              using push_back = type_list<T>;
00859
00860
              template <typename U>
00861
              using concat = U;
00862
00863
              // TODO(jewave): assert index == 0
              template <typename T, size_t index>
using insert = type_list<T>;
00864
00865
00866
00867 } // namespace aerobus
00868
00869 // i16
00870 #ifdef WITH_CUDA_FP16
00871 // i16
00872 namespace aerobus {
00874
          struct i16 {
00875
             using inner_type = int16_t;
00878
              template<int16_t x>
00879
              struct val {
00881
                  using enclosing_type = i16;
                  static constexpr int16_t v = x;
00883
00884
00887
                  template<typename valueType>
00888
                  static constexpr INLINED DEVICE valueType get() {
00889
                      return internal::template int16_convert_helper<valueType, x>::value();
00890
00891
00893
                  using is zero t = std::bool constant<x == 0>;
00894
00896
                  static std::string to_string() {
00897
                     return std::to_string(x);
00898
                  }
00899
              };
00900
00902
              using zero = val<0>;
00904
              using one = val<1>;
00906
              static constexpr bool is_field = false;
00908
              static constexpr bool is_euclidean_domain = true;
00911
              template<auto x>
00912
              using inject constant t = val<static cast<int16 t>(x)>;
00913
00914
              template<typename v>
00915
              using inject_ring_t = v;
00916
           private:
00917
              template<typename v1, typename v2>
00918
00919
              struct add {
00920
                  using type = val<v1::v + v2::v>;
00921
00922
00923
              template<typename v1, typename v2>
00924
              struct sub {
                  using type = val<v1::v - v2::v>;
00925
00926
00927
00928
              template<typename v1, typename v2>
00929
              struct mul {
                  using type = val<v1::v* v2::v>;
00930
00931
              };
```

```
00932
00933
              template<typename v1, typename v2>
              struct div {
00934
                 using type = val<v1::v / v2::v>;
00935
00936
00937
00938
              template<typename v1, typename v2>
00939
              struct remainder {
00940
                 using type = val<v1::v % v2::v>;
00941
00942
00943
              template<typename v1, typename v2>
00944
              struct qt {
00945
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00946
00947
00948
              template<typename v1, typename v2>
00949
              struct lt {
00950
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00951
00952
00953
              template<typename v1, typename v2>
00954
              struct eq {
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00955
00956
00957
              template<typename v1>
00958
00959
              struct pos {
00960
                 using type = std::bool_constant<(v1::v > 0)>;
00961
00962
00963
           public:
00968
              template<typename v1, typename v2>
00969
              using add_t = typename add<v1, v2>::type;
00970
00975
              template<typename v1, typename v2>
00976
              using sub_t = typename sub<v1, v2>::type;
00977
00982
              template<typename v1, typename v2>
00983
              using mul_t = typename mul<v1, v2>::type;
00984
00989
              template<typename v1, typename v2>
00990
              using div t = typename div<v1, v2>::type;
00991
              template<typename v1, typename v2>
00996
00997
              using mod_t = typename remainder<v1, v2>::type;
00998
01003
              template<typename v1, typename v2>
01004
              using gt_t = typename gt<v1, v2>::type;
01005
01010
              template<typename v1, typename v2>
01011
              using lt_t = typename lt<v1, v2>::type;
01012
01017
              template<typename v1, typename v2>
01018
              using eq_t = typename eq<v1, v2>::type;
01019
              template<typename v1, typename v2>
01024
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01025
01030
              template<typename v1, typename v2> \,
              using gcd_t = gcd_t < i16, v1, v2>;
01031
01032
01036
              template<typename v>
01037
              using pos_t = typename pos<v>::type;
01038
01042
              template<typename v>
01043
              static constexpr bool pos_v = pos_t<v>::value;
01044
          };
01045 } // namespace aerobus
01046 #endif
01047
01048 // i32
01049 namespace aerobus {
         struct i32 {
01051
              using inner_type = int32_t;
01052
01055
              template<int32_t x>
01056
              struct val {
01058
                 using enclosing_type = i32;
01060
                  static constexpr int32_t v = x;
01061
01064
                  template<typename valueType>
01065
                  static constexpr DEVICE valueType get() {
01066
                      return static_cast<valueType>(x);
01067
                  }
01068
01070
                  using is_zero_t = std::bool_constant<x == 0>;
01071
```

```
static std::string to_string() {
                     return std::to_string(x);
01074
01075
                  }
01076
             };
01077
01079
              using zero = val<0>;
              using one = val<1>;
01081
01083
              static constexpr bool is_field = false;
01085
              static constexpr bool is_euclidean_domain = true;
01088
              template<auto x>
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01089
01090
01091
              template<typename v>
01092
             using inject_ring_t = v;
01093
          private:
01094
              template<typename v1, typename v2>
01095
01096
              struct add {
01097
                 using type = val<v1::v + v2::v>;
01098
              };
01099
01100
              template<typename v1, typename v2>
01101
              struct sub {
                  using type = val<v1::v - v2::v>;
01102
01103
01104
01105
              template<typename v1, typename v2>
01106
              struct mul {
                 using type = val<v1::v* v2::v>;
01107
01108
01109
01110
              template<typename v1, typename v2>
01111
01112
                 using type = val<v1::v / v2::v>;
01113
01114
              template<typename v1, typename v2>
01115
01116
              struct remainder {
01117
                 using type = val<v1::v % v2::v>;
01118
01119
              template<typename v1, typename v2>
01120
01121
              struct qt {
01122
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01123
01124
01125
              template<typename v1, typename v2>
01126
              struct lt {
                 using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01127
01128
01129
01130
              template<typename v1, typename v2>
01131
              struct eq {
01132
                 using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01133
01134
01135
              template<typename v1>
01136
              struct pos {
01137
                 using type = std::bool_constant<(v1::v > 0)>;
01138
01139
01140
          public:
01145
              template<typename v1, typename v2>
01146
              using add_t = typename add<v1, v2>::type;
01147
01152
              template<typename v1, typename v2>
01153
              using sub_t = typename sub<v1, v2>::type;
01154
01159
              template<typename v1, typename v2>
01160
              using mul_t = typename mul<v1, v2>::type;
01161
01166
              template<typename v1, typename v2>
01167
              using div_t = typename div<v1, v2>::type;
01168
01173
              template<typename v1, typename v2>
01174
              using mod_t = typename remainder<v1, v2>::type;
01175
01180
              template<typename v1, typename v2>
01181
              using gt_t = typename gt<v1, v2>::type;
01182
              template<typename v1, typename v2>
01187
              using lt_t = typename lt<v1, v2>::type;
01188
01189
01194
              template<typename v1, typename v2>
01195
              using eq_t = typename eq<v1, v2>::type;
01196
01200
              template<tvpename v1, tvpename v2>
```

```
01201
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01202
01207
              template<typename v1, typename v2>
01208
              using gcd_t = gcd_t < i32, v1, v2>;
01209
              template<typename v>
01213
01214
             using pos_t = typename pos<v>::type;
01215
01219
              template<typename v>
01220
             static constexpr bool pos_v = pos_t<v>::value;
01221
          };
01222 } // namespace aerobus
01223
01224 // i64
01225 namespace aerobus {
01227
         struct i64 {
              using inner_type = int64_t;
01229
              template<int64_t x>
01232
              struct val {
01235
                  using inner_type = int32_t;
01237
                  using enclosing_type = i64;
01239
                  static constexpr int64_t v = x;
01240
01243
                  template<typename valueType>
01244
                  static constexpr INLINED DEVICE valueType get() {
01245
                    return static_cast<valueType>(x);
01246
01247
                  using is_zero_t = std::bool_constant<x == 0>;
01249
01250
01252
                  static std::string to string() {
01253
                      return std::to_string(x);
01254
01255
              };
01256
01259
              template<auto x>
01260
              using inject_constant_t = val<static_cast<int64_t>(x)>;
01261
01266
              template<typename v>
01267
              using inject_ring_t = v;
01268
01270
              using zero = val<0>;
             using one = val<1>;
01272
01274
              static constexpr bool is_field = false;
01276
             static constexpr bool is_euclidean_domain = true;
01277
01278
              template<typename v1, typename v2>
01279
01280
              struct add {
                 using type = val<v1::v + v2::v>;
01281
01282
              };
01283
01284
              template<typename v1, typename v2>
01285
              struct sub {
01286
                  using type = val<v1::v - v2::v>;
01287
              };
01288
01289
              template<typename v1, typename v2>
01290
              struct mul {
01291
                  using type = val<v1::v* v2::v>;
01292
01293
01294
              template<typename v1, typename v2>
01295
              struct div {
01296
                  using type = val<v1::v / v2::v>;
01297
01298
01299
              template<typename v1, typename v2>
01300
              struct remainder {
                 using type = val<v1::v% v2::v>;
01301
01302
01303
01304
              template<typename v1, typename v2>
01305
              struct qt {
01306
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01307
01308
01309
              template<typename v1, typename v2>
              struct lt {
01310
                 using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01311
01312
01313
01314
              template<typename v1, typename v2>
              struct eq {
01315
01316
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01317
              };
01318
```

```
01319
              template<typename v>
01320
              struct pos {
01321
                  using type = std::bool_constant<(v::v > 0)>;
01322
01323
01324
           public:
01328
              template<typename v1, typename v2>
01329
              using add_t = typename add<v1, v2>::type;
01330
01334
              template<typename v1, typename v2>
              using sub_t = typename sub<v1, v2>::type;
01335
01336
01340
              template<typename v1, typename v2>
01341
              using mul_t = typename mul<v1, v2>::type;
01342
01347
              template<typename v1, typename v2>
01348
              using div_t = typename div<v1, v2>::type;
01349
01353
              template<typename v1, typename v2>
01354
              using mod_t = typename remainder<v1, v2>::type;
01355
01360
              template<typename v1, typename v2>
01361
              using gt_t = typename gt<v1, v2>::type;
01362
01367
              template<typename v1, typename v2>
01368
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01369
01374
              template<typename v1, typename v2>
01375
              using lt_t = typename lt<v1, v2>::type;
01376
              template<typename v1, typename v2>
static constexpr bool lt_v = lt_t<v1, v2>::value;
01381
01382
01383
01388
              template<typename v1, typename v2>
01389
              using eq_t = typename eq<v1, v2>::type;
01390
              template<typename v1, typename v2>
static constexpr bool eq_v = eq_t<v1, v2>::value;
01395
01396
01397
01402
              template<typename v1, typename v2>
01403
              using gcd_t = gcd_t < i64, v1, v2>;
01404
01408
              template<typename v>
01409
              using pos_t = typename pos<v>::type;
01410
01414
              template<typename v>
01415
              static constexpr bool pos_v = pos_t<v>::value;
01416
          };
01417
01419
          template<>
          struct Embed<i32, i64> {
01420
01423
              template<typename val>
01424
              using type = i64::val<static_cast<int64_t>(val::v)>;
01425
01426 } // namespace aerobus
01427
01428 // z/pz
01429 namespace aerobus {
01435
         template<int32_t p>
01436
          struct zpz {
01438
              using inner_type = int32_t;
01439
01442
              template<int32_t x>
01443
              struct val {
01445
                  using enclosing_type = zpz;
01447
                  static constexpr int32_t v = x % p;
01448
                  template<typename valueType>
static constexpr INLINED DEVICE valueType get() {
01451
01452
01453
                       return static_cast<valueType>(x % p);
01454
01455
01457
                  using is_zero_t = std::bool_constant<v == 0>;
01458
                  static constexpr bool is_zero_v = v == 0;
01460
01461
01464
                   static std::string to_string() {
01465
                      return std::to_string(x % p);
01466
                   }
01467
              }:
01468
01471
              template<auto x>
01472
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01473
01475
              using zero = val<0>;
01476
01478
              using one = val<1>:
```

```
01479
01481
              static constexpr bool is_field = is_prime::value;
01482
01484
              static constexpr bool is euclidean domain = true;
01485
01486
           private:
01487
              template<typename v1, typename v2>
01488
              struct add {
01489
                 using type = val<(v1::v + v2::v) % p>;
01490
01491
01492
              template<typename v1, typename v2>
01493
              struct sub {
                  using type = val<(v1::v - v2::v) % p>;
01494
01495
01496
01497
              template<typename v1, typename v2>
01498
              struct mul {
01499
                  using type = val<(v1::v* v2::v) % p>;
01500
01501
01502
              template<typename v1, typename v2>
01503
              struct div {
                  using type = val<(v1::v% p) / (v2::v % p)>;
01504
01505
01506
01507
              template<typename v1, typename v2>
01508
              struct remainder {
01509
                  using type = val<(v1::v% v2::v) % p>;
01510
01511
01512
              template<typename v1, typename v2>
01513
01514
                  using type = std::conditional_t < (v1::v% p > v2::v% p), std::true_type, std::false_type>;
01515
01516
01517
              template<typename v1, typename v2>
              struct lt {
01519
                  using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01520
01521
01522
              template<typename v1, typename v2>
01523
              struct eq {
01524
                  using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01525
01526
01527
              template<typename v1>
01528
              struct pos {
01529
                  using type = std::bool_constant<(v1::v > 0)>;
01530
01532
           public:
01536
              template<typename v1, typename v2>
01537
              using add_t = typename add<v1, v2>::type;
01538
01542
              template<typename v1, typename v2>
              using sub_t = typename sub<v1, v2>::type;
01544
01548
              template<typename v1, typename v2>
              using mul_t = typename mul<v1, v2>::type;
01549
01550
01554
              template<typename v1, typename v2>
              using div_t = typename div<v1, v2>::type;
01556
01560
              template<typename v1, typename v2>
01561
              using mod_t = typename remainder<v1, v2>::type;
01562
01566
              template<typename v1, typename v2>
01567
              using gt_t = typename gt<v1, v2>::type;
01572
              template<typename v1, typename v2>
01573
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01574
01578
              template<typename v1, typename v2>
01579
              using lt_t = typename lt<v1, v2>::type;
01580
01584
              template<typename v1, typename v2>
01585
              static constexpr bool lt_v = lt_t<v1, v2>::value;
01586
01590
              template<typename v1, typename v2>
01591
              using eq_t = typename eq<v1, v2>::type;
01592
01596
              template<typename v1, typename v2>
01597
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01598
              template<typename v1, typename v2>
using gcd_t = gcd_t<i32, v1, v2>;
01602
01603
```

```
template<typename v1>
01607
01608
              using pos_t = typename pos<v1>::type;
01609
01612
              template<typename v>
             static constexpr bool pos_v = pos_t<v>::value;
01613
01614
01615
01618
          template<int32_t x>
01619
          struct Embed<zpz<x>, i32> {
              template <typename val>
01622
01623
              using type = i32::val<val::v>;
01624
01625 } // namespace aerobus
01626
01627 // polynomial
01628 namespace aerobus {
         // coeffN x^N + ...
01629
          template<typename Ring>
01634
         requires IsEuclideanDomain<Ring>
01635
01636
         struct polynomial {
01637
              static constexpr bool is_field = false;
              static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01638
01639
01642
              template<typename P>
01643
              struct horner_reduction_t {
01644
                  template<size_t index, size_t stop>
01645
                  struct inner {
01646
                      template<typename accum, typename x>
                      using type = typename horner_reduction_t<P>::template inner<index + 1, stop>
01647
01648
                          ::template type<
01649
                              typename Ring::template add_t<</pre>
01650
                                  typename Ring::template mul_t<x, accum>,
01651
                                  typename P::template coeff_at_t<P::degree - index>
01652
                              >, x>;
01653
                  };
01654
01655
                  template<size_t stop>
01656
                  struct inner<stop, stop> {
01657
                     template<typename accum, typename x>
01658
                      using type = accum;
01659
                  };
01660
             }:
01661
01665
              template<typename coeffN, typename... coeffs>
01666
              struct val {
01668
                  using ring_type = Ring;
                  using enclosing_type = polynomial<Ring>;
static constexpr size_t degree = sizeof...(coeffs);
01670
01672
01674
                  using aN = coeffN;
01676
                  using strip = val<coeffs...>;
01678
                  using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01680
                  static constexpr bool is_zero_v = is_zero_t::value;
01681
01682
               private:
                  template<size_t index, typename E = void>
01683
                  struct coeff_at {};
01684
01685
01686
                  template<size_t index>
                  01687
                      using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01688
01689
                  };
01690
01691
                  template<size_t index>
01692
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))» {
01693
                      using type = typename Ring::zero;
01694
                  };
01695
01696
               public:
01699
                  template<size_t index>
01700
                  using coeff_at_t = typename coeff_at<index>::type;
01701
01704
                  static std::string to_string() {
01705
                      return string_helper<coeffN, coeffs...>::func();
01706
01707
01712
                  template<typename arithmeticType>
01713
                  static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
                      #ifdef WITH CUDA FP16
01714
                      arithmeticType start;
01715
01716
                      if constexpr (std::is_same_v<arithmeticType, __half2>) {
                          start = \underline{\underline{\hspace{0.5cm}}}half2(0, 0);
01718
                      } else {
01719
                          start = static_cast<arithmeticType>(0);
01720
01721
                      #else
01722
                      arithmeticType start = static cast<arithmeticType>(0);
```

```
#endif
01724
                      return horner_evaluation<arithmeticType, val>
01725
                               ::template inner<0, degree +
01726
                              ::func(start, x);
01727
                  }
01728
01741
                  template<typename arithmeticType>
01742
                  static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01743
                     return compensated_horner<arithmeticType, val>::func(x);
01744
01745
01746
                  template<typename x>
01747
                  using value_at_t = horner_reduction_t<val>
01748
                      ::template inner<0, degree + 1>
01749
                       ::template type<typename Ring::zero, x>;
01750
              };
01751
01754
              template<typename coeffN>
01755
              struct val<coeffN> {
                  using ring_type = Ring;
01757
01759
                  using enclosing_type = polynomial<Ring>;
01761
                  static constexpr size_t degree = 0;
01762
                  using aN = coeffN;
01763
                  using strip = val<coeffN>;
01764
                  using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01765
01766
                  static constexpr bool is_zero_v = is_zero_t::value;
01767
01768
                  template<size_t index, typename E = void>
01769
                  struct coeff_at {};
01770
                  template<size_t index>
01772
                  struct coeff_at<index, std::enable_if_t<(index == 0)» {</pre>
01773
                      using type = aN;
01774
01775
01776
                  template<size t index>
01777
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01778
                      using type = typename Ring::zero;
01779
01780
01781
                  template<size_t index>
01782
                  using coeff_at_t = typename coeff_at<index>::type;
01783
01784
                  static std::string to_string() {
01785
                      return string_helper<coeffN>::func();
01786
01787
01788
                  template<tvpename arithmeticTvpe>
01789
                  static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01790
                      return coeffN::template get<arithmeticType>();
01791
01792
                  template<typename arithmeticType>
static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01793
01794
01795
                      return coeffN::template get<arithmeticType>();
01796
01797
01798
                  template<typename x>
01799
                  using value_at_t = coeffN;
01800
              }:
01801
01803
              using zero = val<typename Ring::zero>;
              using one = val<typename Ring::one>;
01805
01807
              using X = val<typename Ring::one, typename Ring::zero>;
01808
           private:
01809
              template<typename P, typename E = void>
01810
01811
              struct simplify:
01813
              template <typename P1, typename P2, typename I>
01814
              struct add_low;
01815
              template<typename P1, typename P2>
01816
01817
              struct add {
01818
                  using type = typename simplify<typename add_low<
01819
                  P1,
01820
                  P2,
01821
                  internal::make_index_sequence_reverse<
                  std::max(P1::degree, P2::degree) + 1
01822
01823
                  »::type>::type;
01824
              };
01825
01826
              template <typename P1, typename P2, typename I>
01827
              struct sub_low;
01828
01829
              template <typename P1, typename P2, typename I>
```

```
struct mul_low;
01831
01832
               template<typename v1, typename v2>
01833
                struct mul {
01834
                        using type = typename mul_low<
01835
                             v1.
01836
                             v2,
01837
                             internal::make_index_sequence_reverse<
01838
                             v1::degree + v2::degree + 1
01839
                             »::type;
01840
               };
01841
01842
               template<typename coeff, size_t deg>
01843
               struct monomial;
01844
01845
               template<typename v, typename E = void>
01846
               struct derive_helper {};
01847
01848
               template<typename v>
01849
               struct derive_helper<v, std::enable_if_t<v::degree == 0» {</pre>
01850
                    using type = zero;
01851
01852
               template<typename v>
01853
01854
               struct derive_helper<v, std::enable_if_t<v::degree != 0» {
01855
                   using type = typename add<
01856
                         typename derive_helper<typename simplify<typename v::strip>::type>::type,
01857
                         typename monomial<
01858
                             typename Ring::template mul_t<</pre>
01859
                                  typename v::aN,
01860
                                 typename Ring::template inject_constant_t<(v::degree)>
01861
01862
                             v::degree - 1
01863
                        >::type
01864
                    >::type;
01865
               };
01866
01867
               template<typename v1, typename v2, typename E = void>
01868
               struct eq_helper {};
01869
01870
               template<typename v1, typename v2> \,
               struct_eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
01871
01872
                    using type = std::false_type;
01873
01874
01875
               template<typename v1, typename v2>
struct eq_helper<v1, v2, std::enable_if_t<
   v1::degree == v2::degree &&
   (v1::degree != 0 || v2::degree != 0) &&</pre>
01876
01877
01878
01879
01880
                    std::is_same<
01881
                    typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01882
                    std::false_type
01883
                    >::value
01884
01885
               > {
01886
                    using type = std::false_type;
01887
               };
01888
01889
               template<typename v1, typename v2>
               struct eq_helper<v1, v2, std::enable_if_t<
    v1::degree == v2::degree &&</pre>
01890
01891
01892
                    (v1::degree != 0 || v2::degree != 0) &&
01893
                    std::is_same<
01894
                    typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01895
                    std::true_type
01896
                    >::value
01897
               » {
01898
                    using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
               };
01900
01901
                template<typename v1, typename v2>
               struct eq_helper<v1, v2, std::enable_if_t<
   v1::degree == v2::degree &&</pre>
01902
01903
01904
                    (v1::degree == 0)
01905
               » {
01906
                    using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01907
01908
01909
               template<typename v1, typename v2, typename E = void>
01910
               struct lt_helper {};
01911
               template<typename v1, typename v2>
struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {</pre>
01912
01913
01914
                    using type = std::true_type;
01915
               };
01916
```

```
template<typename v1, typename v2>
01918
               struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01919
                   using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01920
01921
               template<typename v1, typename v2>
01922
               struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
01923
01924
                   using type = std::false_type;
01925
01926
01927
               template<typename v1, typename v2, typename E = void>
01928
               struct gt_helper {};
01929
               01930
01931
01932
                  using type = std::true_type;
01933
               };
01934
01935
               template<typename v1, typename v2>
               struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01936
01937
                   using type = std::false_type;
01938
01939
               template<typename v1, typename v2>
struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {</pre>
01940
01941
01942
                 using type = std::false_type;
01943
01944
               // when high power is zero : strip
01945
01946
               template<typename P>
               struct simplify<P, std::enable_if_t<
01947
01948
                   std::is_same<
01949
                   typename Ring::zero,
01950
                   typename P::aN
01951
                  >::value && (P::degree > 0)
01952
               » {
01953
                   using type = typename simplify<typename P::strip>::type;
01954
               };
01955
01956
               // otherwise : do nothing
01957
               template<typename P>
01958
               struct simplify<P, std::enable_if_t<
01959
                  !std::is_same<
typename Ring::zero,
01960
01961
                   typename P::aN
01962
                   >::value && (P::degree > 0)
01963
01964
                   using type = P;
01965
               };
01966
01967
               // do not simplify constants
01968
               template<typename P>
01969
               struct simplify<P, std::enable_if_t<P::degree == 0» {</pre>
01970
                   using type = P;
01971
01972
01973
               // addition at
01974
               template<typename P1, typename P2, size_t index>
01975
               struct add_at {
01976
                   using type =
01977
                       typename Ring::template add t<
01978
                           typename P1::template coeff at t<index>,
01979
                           typename P2::template coeff_at_t<index>;
01980
01981
01982
               template<typename P1, typename P2, size_t index>
01983
               using add_at_t = typename add_at<P1, P2, index>::type;
01984
               template<typename P1, typename P2, std::size_t... I>
struct add_low<P1, P2, std::index_sequence<I...» {</pre>
01985
01987
                  using type = val<add_at_t<P1, P2, I>...>;
01988
01989
               // substraction at
01990
               template<typename P1, typename P2, size_t index>
01991
01992
               struct sub_at {
01993
                   using type =
01994
                       typename Ring::template sub_t<</pre>
01995
                           typename P1::template coeff_at_t<index>,
                           typename P2::template coeff_at_t<index»;
01996
01997
01998
01999
               template<typename P1, typename P2, size_t index>
02000
               using sub_at_t = typename sub_at<P1, P2, index>::type;
02001
               template<typename P1, typename P2, std::size_t... I>
struct sub_low<P1, P2, std::index_sequence<I...» {</pre>
02002
02003
```

```
using type = val<sub_at_t<P1, P2, I>...>;
02005
02006
02007
              template<typename P1, typename P2>
02008
              struct sub {
02009
                  using type = typename simplify<typename sub low<
02010
02011
                  P2,
02012
                  internal::make_index_sequence_reverse<</pre>
02013
                  std::max(P1::degree, P2::degree) + 1
02014
                  »::type>::type;
02015
              };
02016
02017
              // multiplication at
02018
              template<typename v1, typename v2, size_t k, size_t index, size_t stop>
02019
              struct mul_at_loop_helper {
02020
                  using type = typename Ring::template add_t<
                      typename Ring::template mul_t<
02021
                       typename v1::template coeff_at_t<index>,
02022
02023
                      typename v2::template coeff_at_t<k - index>
02024
02025
                       typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
02026
                  >;
02027
              };
02028
02029
              template<typename v1, typename v2, size_t k, size_t stop>
02030
              struct mul_at_loop_helper<v1, v2, k, stop, stop> {
02031
                  using type = typename Ring::template mul_t<</pre>
02032
                       typename v1::template coeff_at_t<stop>,
                       typename v2::template coeff_at_t<0»;</pre>
02033
02034
              };
02035
02036
              template <typename v1, typename v2, size_t k, typename E = void>
02037
              struct mul_at {};
02038
              template<typename v1, typename v2, size_t k>
02039
              struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)» {
    using type = typename Ring::zero;
02040
02041
02042
02043
02044
              template<typename v1, typename v2, size_t k>
              02045
02046
                  using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
02047
02048
02049
              template<typename P1, typename P2, size_t index>
02050
              using mul_at_t = typename mul_at<P1, P2, index>::type;
02051
02052
              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>...>;
02053
02054
02055
02056
              // division helper
02057
02058
              template< typename A, typename B, typename Q, typename R, typename E = void>
02059
              struct div helper {};
02060
02061
              template<typename A, typename B, typename Q, typename R>
02062
              struct div_helper<A, B, Q, R, std::enable_if_t<
02063
                  (R::degree < B::degree) ||
                   (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
02064
                  using q_type = Q;
02065
02066
                  using mod_type = R;
02067
                  using gcd_type = B;
02068
              };
02069
02070
              template<typename A, typename B, typename Q, typename R>
struct div_helper<A, B, Q, R, std::enable_if_t<</pre>
02071
02072
                   (R::degree >= B::degree) &&
02073
                   !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
02074
               private: // NOLINT
02075
                  using rN = typename R::aN;
                  using bN = typename B::aN;
02076
                  using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
02077
     B::degree>::type;
02078
                  using rr = typename sub<R, typename mul<pT, B>::type>::type;
02079
                  using qq = typename add<Q, pT>::type;
02080
               public:
02081
                  using q_type = typename div_helper<A, B, qq, rr>::q_type;
02082
02083
                  using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
02084
                  using gcd_type = rr;
02085
02086
02087
              template<typename A, typename B>
02088
              struct div {
02089
                  static assert (Ring::is euclidean domain, "cannot divide in that type of Ring");
```

```
using q_type = typename div_helper<A, B, zero, A>::q_type;
02091
                  using m_type = typename div_helper<A, B, zero, A>::mod_type;
02092
              };
02093
02094
              template<typename P>
02095
              struct make unit {
02096
                  using type = typename div<P, val<typename P::aN»::q_type;
02097
02098
02099
              template<typename coeff, size_t deg>
02100
              struct monomial {
02101
                 using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
02102
02103
02104
               template<typename coeff>
02105
              struct monomial<coeff, 0> {
02106
                  using type = val<coeff>;
02107
02108
02109
              template<typename arithmeticType, typename P>
02110
              struct horner_evaluation {
02111
                  template<size_t index, size_t stop>
02112
                  struct inner {
                       static constexpr DEVICE INLINED arithmeticType func(
02113
                           const arithmeticType& accum, const arithmeticType& x) {
02114
                           return horner_evaluation<arithmeticType, P>::template inner<index + 1,</pre>
02115
      stop>::func(
02116
                               internal::fma_helper<arithmeticType>::eval(
02117
02118
                                   accum,
                                   P::template coeff_at_t<P::degree - index>::template
02119
      get<arithmeticType>()), x);
02120
02121
                  };
02122
02123
                  template<size_t stop>
                  struct inner<stop, stop> {
02124
                      static constexpr DEVICE INLINED arithmeticType func(
02126
                          const arithmeticType& accum, const arithmeticType& x) {
02127
                           return accum;
02128
02129
                  };
02130
              }:
02131
02132
              template<typename arithmeticType, typename P>
02133
               struct compensated_horner {
02134
                 template<int64_t index, int ghost>
02135
                  struct EFTHorner {
                       static INLINED DEVICE void func(
02136
                               \verb|arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType|\\
02137
      *r) {
02138
                           arithmeticType p;
02139
                           internal::two_prod(*r, x, &p, pi + P::degree - index - 1);
02140
                           constexpr arithmeticType coeff = P::template coeff_at_t<index>::template
      get<arithmeticType>();
02141
                           internal::two sum<arithmeticType>(
02142
                              p, coeff,
                               r, sigma + P::degree - index - 1);
02143
                           EFTHorner<index - 1, ghost>::func(x, pi, sigma, r);
02144
02145
                       }
02146
                  }:
02147
02148
                   template<int ghost>
02149
                  struct EFTHorner<-1, ghost> {
02150
                       static INLINED DEVICE void func(
02151
                               \verb|arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType | \\
      *r) {
02152
                       }
02153
                  };
02155
                   static INLINED DEVICE arithmeticType func(arithmeticType x) {
02156
                       arithmeticType pi[P::degree], sigma[P::degree];
02157
                       \verb|arithmeticType r = P::template coeff_at_t < P::degree > ::template get < arithmeticType > (); \\
02158
                       EFTHorner<P::degree - 1, 0>::func(x, pi, sigma, &r);
                       arithmeticType c = internal::horner<arithmeticType, P::degree - 1>(pi, sigma, x);
02159
02160
                       return r + c;
02161
02162
02163
02164
              template<typename coeff, typename... coeffs>
02165
              struct string_helper {
02166
                  static std::string func() {
                       std::string tail = string_helper<coeffs...>::func();
std::string result = "";
02167
02168
02169
                       if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
                       return tail;
} else if (Ring::template eg_t<coeff, typename Ring::one>::value) {
02170
02171
```

```
02172
                          if (sizeof...(coeffs) == 1) {
02173
                              result += "x";
                          } else {
02174
                              result += "x^" + std::to_string(sizeof...(coeffs));
02175
02176
02177
                      } else {
02178
                          if (sizeof...(coeffs) == 1) {
02179
                              result += coeff::to_string() + " x";
                          } else {
02180
                              02181
02182
02183
02184
                      }
02185
02186
                      if (!tail.empty()) {
                          if (tail.at(0) != '-') {
    result += " + " + tail;
02187
02188
02189
                          } else {
                              result += " - " + tail.substr(1);
02190
02191
02192
02193
02194
                      return result;
02195
                  }
02196
              };
02197
02198
              template<typename coeff>
02199
              struct string_helper<coeff> {
02200
                  static std::string func() {
02201
                      if (!std::is_same<coeff, typename Ring::zero>::value) {
02202
                          return coeff::to_string();
02203
                      } else {
02204
                         return "";
02205
02206
              };
02207
02208
02209
          public:
02212
              template<typename P>
02213
              using simplify_t = typename simplify<P>::type;
02214
02218
              template<typename v1, typename v2>
02219
              using add t = typename add<v1, v2>::type;
02220
02224
              template<typename v1, typename v2>
02225
                         = typename sub<v1, v2>::type;
02226
02230
              template<typename v1, typename v2>
02231
              using mul_t = typename mul<v1, v2>::type;
02232
              template<typename v1, typename v2>
02237
              using eq_t = typename eq_helper<v1, v2>::type;
02238
02242
              template<typename v1, typename v2>
02243
              using lt_t = typename lt_helper<v1, v2>::type;
02244
02248
              template<typename v1, typename v2>
02249
              using gt_t = typename gt_helper<v1, v2>::type;
02250
02254
              template<typename v1, typename v2>
02255
              using div_t = typename div<v1, v2>::q_type;
02256
02260
              template<typename v1, typename v2>
02261
              using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
02262
02266
              template<typename coeff, size_t deg>
02267
              using monomial_t = typename monomial<coeff, deg>::type;
02268
02271
              template<tvpename v>
02272
              using derive_t = typename derive_helper<v>::type;
02273
02276
              template < typename v >
02277
              using pos_t = typename Ring::template pos_t<typename v::aN>;
02278
02281
              template<typename v>
02282
              static constexpr bool pos_v = pos_t<v>::value;
02283
02287
              template<typename v1, typename v2>
              using gcd_t = std::conditional_t<
02288
02289
                  Ring::is euclidean_domain,
                  typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
02290
02291
                  void>;
02292
02295
02296
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
02297
02300
              template<tvpename v>
```

```
using inject_ring_t = val<v>;
02302
02303 } // namespace aerobus
02304
02305 // fraction field
02306 namespace aerobus {
         namespace internal {
02308
              template<typename Ring, typename E = void>
02309
              requires IsEuclideanDomain<Ring>
02310
              struct _FractionField {};
02311
02312
             template<tvpename Ring>
02313
              requires IsEuclideanDomain<Ring>
02314
              struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
02316
                 static constexpr bool is_field = true;
02317
                  static constexpr bool is_euclidean_domain = true;
02318
02319
              private:
02320
                 template<typename val1, typename val2, typename E = void>
02321
                  struct to_string_helper {};
02322
02323
                  template<typename val1, typename val2>
02324
                  struct to_string_helper <val1, val2,
02325
                      std::enable if t<
02326
                      Ring::template eq_t<
02327
                      val2, typename Ring::one
02328
                      >::value
02329
02330
                  > {
02331
                      static std::string func() {
02332
                          return val1::to string();
02333
02334
02335
02336
                  template<typename val1, typename val2>
02337
                  struct to_string_helper<val1, val2,
02338
                      std::enable if t<
02339
                      !Ring::template eq_t<
02340
                      val2.
02341
                      typename Ring::one
02342
                      >::value
02343
02344
                  > {
02345
                      static std::string func() {
02346
                         return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
02347
02348
                  };
02349
02350
               public:
02354
                 template<typename val1, typename val2>
02355
                  struct val {
02357
                      using x = val1;
02359
                      using y = val2;
02361
                      using is_zero_t = typename val1::is_zero_t;
02363
                      static constexpr bool is_zero_v = val1::is_zero_t::value;
02364
02366
                      using ring_type = Ring;
02367
                      using enclosing_type = _FractionField<Ring>;
02368
02371
                      static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
02372
02373
                      template<typename valueType, int ghost = 0>
02374
                      struct get_helper {
02375
                        static constexpr INLINED DEVICE valueType get() {
02376
                              return internal::staticcast<valueType, typename
     ring_type::inner_type>::template func<x::v>() /
02377
                                  internal::staticcast<valueType, typename ring_type::inner_type>::template
     func<v::v>();
02378
02379
                      };
02380
02381
                      #ifdef WITH_CUDA_FP16
02382
                      template<int ghost>
02383
                      struct get_helper<__half, ghost> {
                          static constexpr INLINED DEVICE __half get() {
02384
02385
                              return internal::my_float2half_rn(
02386
                                  internal::staticcast<float, typename ring_type::inner_type>::template
     func<x::v>() /
02387
                                  internal::staticcast<float, typename ring_type::inner_type>::template
     func<v::v>());
02388
02389
                      };
02390
02391
                      template<int ghost>
02392
                      struct get_helper<__half2, ghost> {
                          static constexpr INLINED DEVICE __half2 get() {
02393
                              constexpr __half tmp = internal::my_float2half_rn(
02394
```

```
02395
                                   internal::staticcast<float, typename ring_type::inner_type>::template
      func<x::v>() /
02396
                                  internal::staticcast<float, typename ring_type::inner_type>::template
      func<v::v>());
02397
                               return __half2(tmp, tmp);
02398
                          }
02399
                      };
02400
                       #endif
02401
02405
                      template<typename valueType>
                      static constexpr INLINED DEVICE valueType get() {
02406
02407
                          return get_helper<valueType, 0>::get();
02408
02409
02412
                      static std::string to_string() {
02413
                          return to_string_helper<val1, val2>::func();
02414
02415
02420
                      template<typename arithmeticType>
                      static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& v) {
02421
02422
                          return x::eval(v) / y::eval(v);
02423
02424
                  };
02425
02427
                  using zero = val<typename Ring::zero, typename Ring::one>;
                  using one = val<typename Ring::one, typename Ring::one>;
02429
02430
02433
                  template<typename v>
02434
                  using inject_t = val<v, typename Ring::one>;
02435
02438
                  template<auto x>
02439
                  using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
     Ring::one>;
02440
02443
                  template < typename v >
                  using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
02444
02445
                  using ring_type = Ring;
02448
02449
               private:
02450
                  template<typename v, typename E = void>
02451
                  struct simplify {};
02452
02453
02454
                  template<typename v>
02455
                  struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {</pre>
02456
                      using type = typename _FractionField<Ring>::zero;
02457
                  };
02458
02459
                  // x != 0
02460
                  template<typename v>
02461
                  struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value» {
                   private:
02462
02463
                      using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
02464
                      using newx = typename Ring::template div_t<typename v::x, _gcd>;
                      using newy = typename Ring::template div_t<typename v::y, _gcd>;
02465
02466
02467
                      using posx = std::conditional t<
02468
                                           !Ring::template pos_v<newy>,
02469
                                           typename Ring::template sub_t<typename Ring::zero, newx>,
02470
                                           newx>:
02471
                      using posy = std::conditional_t<
02472
                                           !Ring::template pos_v<newy>,
02473
                                           typename Ring::template sub_t<typename Ring::zero, newy>,
02474
                   public:
02475
02476
                      using type = typename _FractionField<Ring>::template val<posx, posy>;
02477
                  };
02478
02479
               public:
02482
                  template<typename v>
02483
                  using simplify_t = typename simplify<v>::type;
02484
02485
                  template<typename v1, typename v2>
02486
02487
                  struct add {
02488
02489
                      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>;
02490
02491
                      using dividend = typename Ring::template add t<a, b>;
                      using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02492
02493
                      using g = typename Ring::template gcd_t<dividend, diviser>;
02494
                   public:
02495
02496
                      using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
     diviser»;
02497
                  };
```

```
template<typename v>
02499
02500
                                 struct pos
02501
                                       using type = std::conditional_t<
                                                \label{eq:condition} $$(\text{Ring::template pos_v<typename v::y>}) \mid | $$
02502
                                                (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
02503
02504
                                               std::true_type,
02505
                                               std::false_type>;
02506
                                } ;
02507
02508
                                template<typename v1, typename v2>
02509
                                 struct sub {
02510
                                  private:
02511
                                        using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02512
                                        using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
                                       using dividend = typename Ring::template sub_t<a, b>;
using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02513
02514
02515
                                        using g = typename Ring::template gcd_t<dividend, diviser>;
02516
02517
                                  public:
                                        using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
02518
          diviser»;
02519
                                 };
02520
02521
                                template<typename v1, typename v2>
02522
                                struct mul {
02523
                                  private:
02524
                                        using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
02525
                                        using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02526
02527
                                  public:
02528
                                       using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
02529
02530
02531
                                 template<typename v1, typename v2, typename E = void>
02532
                                 struct div {}:
02533
                                 template<typename v1, typename v2>
02535
                                 struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
_FractionField<Ring>::zero>::value» {
02537
                                        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>;
02538
02539
02540
                                        using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
02541
02542
02543
02544
                                template<typename v1, typename v2>
02545
                                struct div<v1, v2, std::enable if t<
                                        std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
02547
                                        using type = one;
02548
02549
02550
                                template<typename v1, typename v2>
02551
                                struct eq {
02552
                                       using type = std::conditional_t<
02553
                                                       std::is_same<typename simplify_t<vl>::x, typename simplify_t<v2>::x>::value &&
02554
                                                       std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
                                               std::true_type,
02555
02556
                                               std::false_type>;
02557
                                };
02558
02559
                                 template<typename v1, typename v2, typename E = void>
02560
                                 struct gt;
02561
02562
                                 template<typename v1, typename v2>
                                struct qt<v1, v2, std::enable_if_t<
02563
                                        (eq<v1, v2>::type::value)
02564
02565
02566
                                        using type = std::false_type;
02567
                                 };
02568
02569
                                 template<typename v1, typename v2>
                                 struct gt<v1, v2, std::enable_if_t<
02570
                                        (!eq<v1, v2>::type::value) &&
02571
02572
                                        (!pos<v1>::type::value) && (!pos<v2>::type::value)
02573
02574
                                        using type = typename gt<
                                               typename sub<zero, v1>::type, typename sub<zero, v2>::type
02575
02576
                                        >::type;
02577
                                } ;
02578
02579
                                 template<typename v1, typename v2>
02580
                                 struct gt<v1, v2, std::enable_if_t<
                                        (!eq<v1, v2>::type::value) &&
02581
02582
                                        (pos<v1>::type::value) && (!pos<v2>::type::value)
```

```
02584
                      using type = std::true_type;
02585
                  };
02586
02587
                  template<typename v1, typename v2>
                  struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02588
02589
02590
                       (!pos<v1>::type::value) && (pos<v2>::type::value)
02591
02592
                      using type = std::false_type;
02593
                  };
02594
02595
                  template<typename v1, typename v2>
02596
                  struct gt<v1, v2, std::enable_if_t<
02597
                      (!eq<v1, v2>::type::value) &&
02598
                       (pos<v1>::type::value) && (pos<v2>::type::value)
02599
02600
                      using type = typename Ring::template gt t<
                          typename Ring::template mul_t<v1::x, v2::y>,
02601
02602
                           typename Ring::template mul_t<v2::y, v2::x>
02603
02604
                  };
02605
02606
               public:
02610
                  template<typename v1, typename v2>
                  using add_t = typename add<v1, v2>::type;
02611
02612
02617
                  template<typename v1, typename v2>
02618
                  using mod t = zero;
02619
02624
                  template<typename v1, typename v2>
02625
                  using gcd_t = v1;
02626
02630
                  template<typename v1, typename v2>
02631
                  using sub_t = typename sub<v1, v2>::type;
02632
02636
                  template<typename v1, typename v2>
02637
                  using mul_t = typename mul<v1, v2>::type;
02638
02642
                  template<typename v1, typename v2>
02643
                  using div_t = typename div<v1, v2>::type;
02644
                  template<typename v1, typename v2>
02648
02649
                  using eq_t = typename eq<v1, v2>::type;
02650
02654
                  template<typename v1, typename v2>
02655
                  static constexpr bool eq_v = eq<v1, v2>::type::value;
02656
02660
                  template<typename v1, typename v2>
                  using gt_t = typename gt<v1, v2>::type;
02661
02662
02666
                  template<typename v1, typename v2>
02667
                  static constexpr bool gt_v = gt<v1, v2>::type::value;
02668
02671
                  template<typename v1>
02672
                  using pos_t = typename pos<v1>::type;
02673
02676
                  template<typename v>
02677
                   static constexpr bool pos_v = pos_t<v>::value;
02678
              };
02679
02680
              template<typename Ring, typename E = void>
02681
              requires IsEuclideanDomain<Ring>
02682
              struct FractionFieldImpl {};
02683
02684
              // fraction field of a field is the field itself
02685
              template<typename Field>
              requires IsEuclideanDomain<Field>
02686
02687
              struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {</pre>
02688
                  using type = Field;
02689
                  template<typename v>
02690
                  using inject_t = v;
02691
              };
02692
02693
              // fraction field of a ring is the actual fraction field
02694
              template<typename Ring>
02695
              requires IsEuclideanDomain<Ring>
02696
              struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field> {
02697
                  using type = _FractionField<Ring>;
02698
          } // namespace internal
02699
02700
02703
          template<typename Ring>
02704
          requires IsEuclideanDomain<Ring>
02705
          using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02706
02709
          template<typename Ring>
```

```
02710
          struct Embed<Ring, FractionField<Ring» {</pre>
02713
              template<typename v>
02714
              using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02715
02716 }
        // namespace aerobus
02717
02718
02719 // short names for common types
02720 namespace aerobus {
02724
          template<typename X, typename Y>
02725
          requires IsRing<typename X::enclosing_type> &&
02726
             (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02727
          using add_t = typename X::enclosing_type::template add_t<X, Y>;
02728
02732
          template<typename X, typename Y>
02733
          requires IsRing<typename X::enclosing_type> &&
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02734
02735
          using sub_t = typename X::enclosing_type::template sub_t<X, Y>;
02740
          template<typename X, typename Y>
02741
          requires IsRing<typename X::enclosing_type> &&
02742
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02743
          using mul_t = typename X::enclosing_type::template mul_t<X, Y>;
02744
02748
          template<typename X, typename Y>
02749
          requires IsEuclideanDomain<typename X::enclosing_type> &&
02750
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02751
          using div_t = typename X::enclosing_type::template div_t<X, Y>;
02752
02755
          using g32 = FractionField<i32>;
02756
02759
          using fpq32 = FractionField<polynomial<q32>>;
02760
02763
          using q64 = FractionField<i64>;
02764
          using pi64 = polynomial<i64>;
02766
02767
02769
          using pq64 = polynomial<q64>;
02770
02772
          using fpq64 = FractionField<polynomial<q64>>;
02773
02778
          template<typename Ring, typename v1, typename v2> \,
02779
          using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02780
02787
          template<typename v>
02788
          using embed_int_poly_in_fractions_t =
02789
                  typename Embed<
02790
                      polynomial<typename v::ring_type>,
02791
                      polynomial<FractionField<typename v::ring_type>»::template type<v>;
02792
          template<int64_t p, int64_t q>
using make_q64_t = typename q64::template simplify_t<</pre>
02796
02797
02798
                      typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02799
02803
         02804
02805
02806
02807
          #ifdef WITH_CUDA_FP16
02809
          using q16 = FractionField<i16>;
02810
          template<int16_t p, int16_t q>
using make_q16_t = typename q16::template simplify_t<</pre>
02814
02815
02816
                      typename q16::val<i16::inject_constant_t<p>, i16::inject_constant_t<q>>;
02817
          #endif
02818
02823
          template<typename Ring, typename v1, typename v2>
          using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02824
02829
          template<typename Ring, typename v1, typename v2>
          using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02830
02831
          template<>
02833
02834
          struct Embed<q32, q64> {
02837
             template<typename v>
02838
              using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02839
02840
02844
          template<typename Small, typename Large>
02845
          struct Embed<polynomial<Small>, polynomial<Large» {</pre>
          private:
02846
02847
             template<typename v, typename i>
02848
              struct at_low;
02849
02850
              template<typename v, size_t i>
              struct at_index {
02851
                 using type = typename Embed<Small, Large>::template
02852
      type<typename v::template coeff_at_t<i>>;
```

```
};
02854
02855
              template<typename v, size_t... Is>
02856
              struct at_low<v, std::index_sequence<Is...» {</pre>
02857
                  using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
              };
02858
02859
02860
           public:
02863
             template<typename v>
02864
              using type = typename at_low<v, typename internal::make_index_sequence_reverse<v::degree +
     1»::type;
02865
         };
02866
02870
          template<typename Ring, auto... xs>
02871
          using make_int_polynomial_t = typename polynomial<Ring>::template val<</pre>
02872
                   typename Ring::template inject_constant_t<xs>...>;
02873
02877
          template<typename Ring, auto... xs>
          using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<
02878
                  typename FractionField<Ring>::template inject_constant_t<xs>...>;
02879
02880 } // namespace aerobus
02881
02882 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02883 namespace aerobus {
02884
          namespace internal {
            template<typename T, size_t x, typename E = void>
02885
02886
              struct factorial {};
02887
02888
              template<typename T, size_t x>
02889
              struct factorial<T, x, std::enable_if_t<(x > 0)  {
02890
              private:
02891
                  template<typename, size t, typename>
02892
                   friend struct factorial;
02893
              public:
02894
                  using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
     x - 1>::type>;
02895
                  static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
02896
              };
02897
02898
              template<typename T>
02899
              struct factorial<T, 0> {
02900
              public:
02901
                  using type = typename T::one;
                   static constexpr typename T::inner_type value = type::template get<typename</pre>
     T::inner_type>();
02903
          } // namespace internal
02904
02905
02909
          template<tvpename T, size t i>
02910
          using factorial_t = typename internal::factorial<T, i>::type;
02911
02915
          template<typename T, size_t i>
02916
          inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02917
02918
          namespace internal {
              template<typename T, size_t k, size_t n, typename E = void>
02919
02920
              struct combination_helper {};
02921
02922
              template<typename T, size_t k, size_t n>
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
    using type = typename FractionField<T>::template mul_t<
        typename combination_helper<T, k - 1, n - 1>::type,
02923
02924
02925
02926
                       makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02927
              };
02928
02929
              template<typename T, size_t k, size_t n>
              struct combination_helperTT, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)» {
02930
                  using type = typename combination_helper<T, n - k, n>::type;
02931
02932
              };
02933
02934
              template<typename T, size_t n>
02935
              struct combination_helper<T, 0, n> {
02936
                  using type = typename FractionField<T>::one;
02937
02938
02939
               template<typename T, size_t k, size_t n>
02940
               struct combination {
02941
                  using type = typename internal::combination_helper<T, k, n>::type::x;
                   static constexpr typename T::inner_type value =
02942
                               internal::combination_helper<T, k, n>::type::template get<typename</pre>
02943
     T::inner_type>();
02944
02945
          } // namespace internal
02946
          template<typename T, size_t k, size_t n>
using combination_t = typename internal::combination<T, k, n>::type;
02949
02950
```

```
02951
02956
           template<typename T, size_t k, size_t n>
02957
          inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02958
02959
          namespace internal {
               template<typename T, size_t m>
02960
02961
               struct bernoulli;
02962
02963
               template<typename T, typename accum, size_t k, size_t m>
02964
               struct bernoulli helper {
                   using type = typename bernoulli_helper<
02965
02966
                       Τ.
02967
                       addfractions_t<T,
02968
                           accum,
02969
                           mulfractions_t<T,</pre>
02970
                               makefraction_t<T,
02971
                                    combination_t<T, k, m + 1>,
02972
                                    typename T::one>,
02973
                                typename bernoulli<T, k>::type
02974
02975
                       >,
k + 1,
02976
02977
                       m>::type;
02978
               };
02979
02980
               template<typename T, typename accum, size_t m>
02981
               struct bernoulli_helper<T, accum, m, m> {
02982
                   using type = accum;
02983
02984
02985
02986
02987
               template<typename T, size_t m>
02988
               struct bernoulli {
02989
                   using type = typename FractionField<T>::template mul_t<</pre>
                       typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02990
02991
                       makefraction t<T,
02992
                       typename T::template val<static_cast<typename T::inner_type>(-1)>,
02993
                       typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02994
02995
02996
                   template<typename floatType>
02997
02998
                   static constexpr floatType value = type::template get<floatType>();
02999
              };
03000
03001
               template<typename T>
03002
               struct bernoulli<T, 0> {
                  using type = typename FractionField<T>::one;
03003
03004
03005
                   template<typename floatType>
03006
                   static constexpr floatType value = type::template get<floatType>();
03007
               } ;
03008
          } // namespace internal
03009
03013
          template<typename T, size_t n>
using bernoulli_t = typename internal::bernoulli<T, n>::type;
03014
03015
          template<typename FloatType, typename T, size_t n >
inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
03020
03021
03022
03023
          // bell numbers
03024
          namespace internal {
03025
              template<typename T, size_t n, typename E = void>
03026
               struct bell_helper;
03027
03028
               template <typename T, size_t n>
               struct bell_helper<T, n, std::enable_if_t<(n > 1)» {
03029
03030
                   template<typename accum, size t i, size t stop>
03031
                   struct sum_helper {
03032
                   private:
03033
                       using left = typename T::template mul_t<</pre>
                                    combination_t<T, i, n-1>,
typename bell_helper<T, i>::type>;
03034
03035
03036
                       using new_accum = typename T::template add_t<accum, left>;
03037
                    public:
03038
                       using type = typename sum_helper<new_accum, i+1, stop>::type;
03039
03040
03041
                   template<typename accum, size_t stop>
03042
                   struct sum_helper<accum, stop, stop> {
03043
                       using type = accum;
03044
03045
03046
                   using type = typename sum_helper<typename T::zero, 0, n>::type;
03047
              };
03048
```

```
03049
               template<typename T>
03050
              struct bell_helper<T, 0> {
03051
                   using type = typename T::one;
03052
03053
03054
               template<tvpename T>
03055
              struct bell_helper<T, 1> {
03056
                  using type = typename T::one;
03057
03058
          } // namespace internal
03059
          template<typename T, size_t n>
03063
03064
          using bell_t = typename internal::bell_helper<T, n>::type;
03065
03069
          template<typename T, size_t n>
03070
          static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
03071
03072
          namespace internal {
              template<typename T, int k, typename E = void>
03073
03074
              struct alternate {};
03075
03076
               template<typename T, int k>
03077
               struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
03078
                   using type = typename T::one;
                   static constexpr typename T::inner_type value = type::template get<typename
03079
      T::inner_type>();
03080
              };
03081
               template<typename T, int k> struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
03082
03083
                   using type = typename T::template sub_t<typename T::zero, typename T::one>;
03084
03085
                   static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
03086
03087
          } // namespace internal
03088
03091
          template<typename T, int k>
03092
          using alternate_t = typename internal::alternate<T, k>::type;
03093
03096
          template<typename T, size_t k>
03097
          inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
03098
03099
          namespace internal {
03100
              template<typename T, int n, int k, typename E = void>
03101
              struct stirling_1_helper {};
03102
03103
               template<typename T>
               struct stirling_1_helper<T, 0, 0> {
03104
                  using type = typename T::one;
03105
03106
03107
03108
               template<typename T, int n>
03109
               struct stirling_1_helper<T, n, 0, std::enable_if_t<(n > 0)» {
03110
                  using type = typename T::zero;
03111
03112
03113
               template<typename T, int n>
              struct stirling_1_helper<T, 0, n, std::enable_if_t<(n > 0)» {
03114
03115
                   using type = typename T::zero;
03116
0.3117
              template<typename T, int n, int k> struct stirling_1_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)» { using type = typename T::template sub_t<
03118
03119
03120
03121
                                    typename stirling_1_helper<T, n-1, k-1>::type,
                                    typename T::template mul_t<</pre>
03122
03123
                                         typename T::template inject_constant_t<n-1>,
                                         typename stirling_1_helper<T, n-1, k>::type
03124
03125
03126
03127
           } // namespace internal
03128
          template<typename T, int n, int k>
using stirling_1_signed_t = typename internal::stirling_1_helper<T, n, k>::type;
03133
03134
03135
03140
          template<typename T, int n, int k>
          using stirling_1_unsigned_t = abs_t<typename internal::stirling_1_helper<T, n, k>::type>;
03141
03142
          template<typename T, int n, int k>
static constexpr typename T::inner_type stirling_1_unsigned_v = stirling_1_unsigned_t<T, n, k>::v;
03147
0.3148
03149
03154
          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;
03155
03156
03157
          namespace internal {
               template<typename T, int n, int k, typename E = void>
03158
03159
               struct stirling_2_helper {};
```

```
03160
03161
               template<typename T, int n>
03162
               struct stirling_2_helper<T, n, n, std::enable_if_t<(n >= 0)» {
03163
                  using type = typename T::one;
03164
03165
03166
               template<typename T, int n>
03167
               struct stirling_2_helper<T, n, 0, std::enable_if_t<(n > 0)» {
03168
                  using type = typename T::zero;
03169
03170
03171
               template<typename T, int n>
03172
               struct stirling_2_helper<T, 0, n, std::enable_if_t<(n > 0) > {
03173
                    using type = typename T::zero;
03174
03175
03176
               template<typename T, int n, int k>
               template typename 1, Int N, Int K, struct stirling_1helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0) && (k < n)» { using type = typename T::template add_t<
03177
03178
                                      typename stirling_2_helper<T, n-1, k-1>::type,
03179
03180
                                      typename T::template mul_t<
03181
                                          typename T::template inject_constant_t<k>,
03182
                                          typename stirling_2_helper<T, n-1, k>::type
03183
03184
03185
           } // namespace internal
03186
           template<typename T, int n, int k>
using stirling_2_t = typename internal::stirling_2_helper<T, n, k>::type;
03191
03192
03193
           template<typename T, int n, int k>
static constexpr typename T::inner_type stirling_2_v = stirling_2_t<T, n, k>::v;
03198
03199
03200
03201
           namespace internal {
03202
                template<typename T>
03203
                struct pow_scalar {
03204
                   template<size t p>
                    static constexpr DEVICE INLINED T func(const T& x) { return p == 0 ? static_cast<T>(1) :
03205
03206
                       p % 2 == 0 ? func < p/2 > (x) * func < p/2 > (x) :
03207
                        x * func < p/2 > (x) * func < p/2 > (x);
03208
03209
               };
03210
03211
               template<typename T, typename p, size_t n, typename E = void>
03212
                requires IsEuclideanDomain<T>
03213
                struct pow;
03214
               template<typename T, typename p, size_t n> struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)» {
03215
03216
03217
                   using type = typename T::template mul t<
03218
                        typename pow<T, p, n/2>::type,
03219
                        typename pow<T, p, n/2>::type
03220
03221
               };
03222
03223
               template<typename T, typename p, size_t n>
               struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)» {
03225
                    using type = typename T::template mul_t<
03226
03227
                        typename T::template mul_t<</pre>
                             typename pow<T, p, n/2>::type, typename pow<T, p, n/2>::type
03228
03229
03230
03231
                    >;
03232
               };
03233
           template<typename T, typename p, size_t n>
    struct pow<T, p, n, std::enable_if_t<n == 0» { using type = typename T::one; };
} // namespace internal</pre>
03234
03235
03236
03237
03242
           template<typename T, typename p, size_t n>
03243
           using pow_t = typename internal::pow<T, p, n>::type;
03244
           template<typename T, typename p, size_t n>
03249
03250
           static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
03251
03252
           template<typename T, size_t p>
03253
           static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
      internal::pow_scalar<T>::template func(x); }
03254
03255
           namespace internal {
               template<typename, template<typename, size_t> typename, class>
03257
               struct make_taylor_impl;
03258
03259
               template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
03260
               struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...» {</pre>
                    using type = typename polynomial<FractionField<T>::template val<typename coeff_at<T,
03261
```

```
Is>::type...>;
03262
              };
03263
03264
03269
          template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
03270
          using taylor = typename internal::make_taylor_impl<</pre>
03271
03272
03273
              internal::make_index_sequence_reverse<deg + 1>>::type;
03274
03275
          namespace internal {
03276
              template<typename T, size_t i>
03277
              struct exp coeff {
03278
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03279
03280
03281
              template<typename T, size_t i, typename E = void>
03282
              struct sin_coeff_helper {};
03283
03284
              template<typename T, size_t i>
03285
              struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
                  using type = typename FractionField<T>::zero;
03286
03287
              };
03288
03289
              template<typename T, size_t i>
              struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03290
03291
                  using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03292
03293
              template<typename T, size_t i>
03294
03295
              struct sin coeff {
03296
                 using type = typename sin_coeff_helper<T, i>::type;
03297
03298
03299
              template<typename T, size_t i, typename E = void>
03300
              struct sh_coeff_helper {};
03301
              template<typename T, size_t i>
03302
03303
              struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03304
                 using type = typename FractionField<T>::zero;
03305
03306
              template<typename T, size_t i>
03307
03308
              struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03309
03310
03311
03312
              template<typename T, size_t i>
              struct sh_coeff {
03313
03314
                  using type = typename sh_coeff_helper<T, i>::type;
03315
03316
03317
              template<typename T, size_t i, typename E = void>
03318
              struct cos_coeff_helper {};
03319
03320
              template<typename T, size t i>
              struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03321
03322
                  using type = typename FractionField<T>::zero;
03323
03324
              template<typename T, size_t i>
03325
              struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03326
03327
                  using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03328
03329
03330
              template<typename T, size_t i>
03331
              struct cos_coeff {
03332
                  using type = typename cos_coeff_helper<T, i>::type;
03333
03334
03335
              template<typename T, size_t i, typename E = void>
03336
              struct cosh_coeff_helper {};
03337
              template<tvpename T, size t i>
03338
03339
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03340
                  using type = typename FractionField<T>::zero;
03341
03342
03343
              template<typename T, size_t i>
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03344
03345
03346
03347
03348
              template<typename T, size_t i>
03349
              struct cosh_coeff {
                  using type = typename cosh_coeff_helper<T, i>::type;
03350
03351
              };
```

```
03352
03353
               template<typename T, size_t i>
03354
              struct geom_coeff { using type = typename FractionField<T>::one; };
03355
03356
              template<typename T, size_t i, typename E = void>
03357
03358
              struct atan_coeff_helper;
03359
03360
              template<typename T, size_t i>
03361
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1  {
                  using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;;
03362
03363
03364
03365
              template<typename T, size_t i>
03366
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03367
                  using type = typename FractionField<T>::zero;
03368
              };
03369
03370
              template<typename T, size_t i>
03371
              struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
03372
03373
              template<typename T, size_t i, typename E = void>
03374
              struct asin_coeff_helper;
03375
03376
              template<typename T, size_t i>
03377
              struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03378
                   using type = makefraction_t<T,
03379
                       factorial_t<T, i - 1>,
03380
                       typename T::template mul_t<</pre>
03381
                           typename T::template val<i>,
03382
                           T::template mul_t<
03383
                               pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
03384
                               pow<T, factorial_t<T, i / 2>, 2
03385
03386
                       >
03387
                       »;
03388
              };
03389
03390
               template<typename T, size_t i>
03391
              struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03392
                   using type = typename FractionField<T>::zero;
03393
              }:
03394
03395
              template<typename T, size_t i>
              struct asin_coeff {
03396
                  using type = typename asin_coeff_helper<T, i>::type;
03397
03398
03399
              template<typename T, size_t i>
03400
03401
              struct lnp1_coeff {
03402
                  using type = makefraction_t<T,
03403
                      alternate_t<T, i + 1>,
03404
                       typename T::template val<i>;
03405
              };
03406
03407
              template<typename T>
              struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
03408
03409
03410
               template<typename T, size_t i, typename E = void>
03411
               struct asinh_coeff_helper;
03412
03413
              template<typename T, size_t i>
03414
              struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03415
                  using type = makefraction_t<T,
03416
                       typename T::template mul_t<
                           alternate_t<T, i / 2>,
factorial_t<T, i - 1>
03417
03418
03419
                       >,
03420
                       typename T::template mul_t<
03421
                           typename T::template mul_t<
03422
                                typename T::template val<i>,
                                pow_t<T, factorial_t<T, i / 2>, 2>
03423
03424
                           pow_t<T, typename T::template inject_constant_t<4>, i / 2>
03425
03426
03427
                  >;
03428
03429
03430
               template<typename T, size_t i>
              struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
03431
03432
03433
03434
03435
              template<typename T, size_t i>
03436
              struct asinh_coeff {
03437
                   using type = typename asinh_coeff_helper<T, i>::type;
03438
              };
```

```
template<typename T, size_t i, typename E = void>
03440
03441
               struct atanh_coeff_helper;
03442
03443
               template<typename T, size_t i>
struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03444
                   // 1/i
03445
03446
                    using type = typename FractionField<T>:: template val<
03447
                        typename T::one,
03448
                        typename T::template inject_constant_t<i>;
03449
               };
03450
03451
               template<typename T, size_t i>
03452
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03453
                   using type = typename FractionField<T>::zero;
03454
03455
03456
               template<typename T, size_t i>
               struct atanh_coeff {
03457
03458
                   using type = typename atanh_coeff_helper<T, i>::type;
03459
03460
03461
               template<typename T, size_t i, typename E = void>
03462
               struct tan_coeff_helper;
03463
03464
               template<typename T, size_t i>
03465
               struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {</pre>
03466
                   using type = typename FractionField<T>::zero;
03467
03468
               template<typename T, size_t i>
struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
03469
03470
03471
                   // 4^((i+1)/2)
03472
03473
                   using _4p = typename FractionField<T>::template inject_t<</pre>
                    pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»; // 4^((i+1)/2) - 1
03474
03475
                   using _4pm1 = typename FractionField<T>::template
      sub_t<_4p, typename FractionField<T>::one>;
03477
                   // (-1)^((i-1)/2)
03478
                   using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
03479
                   using dividend = typename FractionField<T>::template mul_t<</pre>
03480
                        altp,
03481
                        FractionField<T>::template mul_t<</pre>
03482
                        _4p,
03483
                        FractionField<T>::template mul_t<</pre>
03484
                        _4pm1,
                        bernoulli_t<T, (i + 1)>
03485
03486
03487
03488
03489
               public:
03490
                   using type = typename FractionField<T>::template div_t<dividend,</pre>
03491
                        typename FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
03492
               };
03493
03494
               template<typename T, size_t i>
03495
               struct tan coeff {
03496
                   using type = typename tan_coeff_helper<T, i>::type;
03497
03498
               template<typename T, size_t i, typename E = void>
03499
03500
               struct tanh_coeff_helper;
03501
03502
               template<typename T, size_t i>
               struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
    using type = typename FractionField<T>::zero;
03503
03504
03505
03506
               template<typename T, size_t i>
03508
               struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
03509
               private:
03510
                   using _4p = typename FractionField<T>::template inject_t<</pre>
                   pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
using _4pm1 = typename FractionField<T>::template
03511
03512
      sub_t<_4p, typename FractionField<T>::one>;
                   using dividend =
03513
03514
                        typename FractionField<T>::template mul_t<</pre>
03515
                            _4p,
typename FractionField<T>::template mul_t<
03516
03517
                                 4pm1,
03518
                                 bernoulli_t<T, (i + 1) >>::type;
03519
               public:
03520
                   using type = typename FractionField<T>::template div_t<dividend,</pre>
03521
                        FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03522
               };
03523
```

```
template<typename T, size_t i>
03525
                      struct tanh coeff {
03526
                             using type = typename tanh_coeff_helper<T, i>::type;
03527
03528
                } // namespace internal
03529
03533
                template<typename Integers, size_t deg>
03534
                using exp = taylor<Integers, internal::exp_coeff, deg>;
03535
               template<typename Integers, size_t deg>
using expm1 = typename polynomial<FractionField<Integers>>::template sub_t
03539
03540
03541
                      exp<Integers, deg>,
03542
                      typename polynomial<FractionField<Integers>>::one>;
03543
03547
                template<typename Integers, size_t deg>
03548
                using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
03549
03553
                template<typename Integers, size_t deg>
                using atan = taylor<Integers, internal::atan_coeff, deg>;
03554
03555
03559
                template<typename Integers, size_t deg>
03560
                using sin = taylor<Integers, internal::sin_coeff, deg>;
03561
03565
                template<typename Integers, size_t deg>
03566
                using sinh = taylor<Integers, internal::sh_coeff, deg>;
03567
03572
                template<typename Integers, size_t deg>
03573
                using cosh = taylor<Integers, internal::cosh_coeff, deg>;
03574
03579
                template<typename Integers, size_t deg>
03580
                using cos = taylor<Integers, internal::cos coeff, deg>;
03581
03586
                template<typename Integers, size_t deg>
03587
                using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
03588
03593
                template<typename Integers, size_t deg>
03594
                using asin = taylor<Integers, internal::asin_coeff, deq>;
03595
03600
                template<typename Integers, size_t deg>
03601
                using asinh = taylor<Integers, internal::asinh_coeff, deg>;
03602
03607
                template<typename Integers, size_t deg>
03608
                using atanh = taylor<Integers, internal::atanh_coeff, deg>;
03609
03614
                template<typename Integers, size_t deg>
03615
                using tan = taylor<Integers, internal::tan_coeff, deg>;
03616
03621
                template<typename Integers, size_t deg>
                using tanh = taylor<Integers, internal::tanh_coeff, deg>;
03622
03623 }
              // namespace aerobus
03624
03625 // continued fractions
03626 namespace aerobus {
03629
                template<int64 t... values>
03630
                struct ContinuedFraction {};
03631
03634
                template<int64 t a0>
03635
                struct ContinuedFraction<a0> {
03637
                      using type = typename q64::template inject_constant_t<a0>;
03639
                      static constexpr double val = static_cast<double>(a0);
03640
                };
03641
03645
                template<int64_t a0, int64_t... rest>
                struct ContinuedFraction<a0, rest...> {
03646
03648
                      using type = q64::template add_t<
03649
                                    typename q64::template inject_constant_t<a0>,
03650
                                    typename q64::template div_t<
03651
                                          typename q64::one,
03652
                                          typename ContinuedFraction<rest...>::type
03653
03654
03656
                      static constexpr double val = type::template get<double>();
03657
03658
               using PI fraction =
03662
         ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1>;
03664
                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>;
03666
               using SQRT2_fraction =
         03668
         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
03669 } // namespace aerobus
03670
03671 // known polynomials
03672 namespace aerobus {
```

```
// CChebyshev
03674
          namespace internal {
03675
              template<int kind, size_t deg, typename I>
              struct chebyshev_helper {
03676
03677
                  using type = typename polynomial<I>::template sub_t<
                      typename polynomial<I>::template mul_t<
03678
                           typename polynomial<I>::template mul_t<
03679
03680
                               typename polynomial<I>::template inject_constant_t<2>,
03681
                               typename polynomial<I>::X>,
03682
                          typename chebyshev_helper<kind, deg - 1, I>::type
03683
                      typename chebyshev_helper<kind, deg - 2, I>::type
03684
03685
                  >;
03686
              };
03687
03688
              template<typename I>
              struct chebyshev_helper<1, 0, I> {
03689
03690
                 using type = typename polynomial<I>::one;
03691
03692
03693
              template<typename I>
03694
              struct chebyshev_helper<1, 1, I> {
03695
                 using type = typename polynomial<I>::X;
03696
03697
03698
              template<typename I>
03699
              struct chebyshev_helper<2, 0, I> {
03700
                  using type = typename polynomial<I>::one;
03701
03702
03703
              template<tvpename I>
03704
              struct chebyshev_helper<2, 1, I> {
03705
                  using type = typename polynomial<I>::template mul_t<
03706
                      typename polynomial<I>::template inject_constant_t<2>,
03707
                      typename polynomial<I>::X>;
03708
              };
03709
          } // namespace internal
03710
03711
          // Laguerre
03712
          namespace internal {
03713
              template<size_t deg, typename I>
03714
              struct laguerre_helper {
03715
                 using Q = FractionField<I>;
                  using PQ = polynomial<Q>;
03716
03717
03718
               private:
03719
                 // Lk = (1 / k) * ((2 * k - 1 - x) * 1km1 - (k - 2) Lkm2)
                  using lnm2 = typename laguerre_helper<deg - 2, I>::type;
03720
                  using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03721
03722
                  // -x + 2k-1
03723
                  using p = typename PQ::template val<
03724
                      typename Q::template inject_constant_t<-1>,
03725
                      typename Q::template inject_constant_t<2 * deg - 1»;</pre>
03726
                  // 1/n
                  using factor = typename PQ::template inject_ring_t<
03727
03728
                      typename Q::template val<typename I::one, typename I::template
     inject_constant_t<deg>>;
03729
03730
               public:
03731
                  using type = typename PQ::template mul_t <</pre>
03732
                      factor.
03733
                      typename PQ::template sub_t<
03734
                          typename PQ::template mul_t<
03735
                              p,
03736
                               1 nm1
03737
03738
                          typename PQ::template mul_t<</pre>
03739
                               typename PQ::template inject_constant_t<deg-1>,
                               lnm2
03740
03741
03742
03743
                  >;
03744
              };
03745
03746
              template<typename I>
03747
              struct laguerre_helper<0, I> {
03748
                 using type = typename polynomial<FractionField<I»::one;
03749
03750
03751
              template<tvpename T>
03752
              struct laguerre_helper<1, I> {
03753
               private:
                  using PQ = polynomial<FractionField<I>;
03754
               public:
03755
03756
                 using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
03757
03758
          } // namespace internal
```

```
03759
03760
03761
          namespace internal {
03762
               template<size_t i, size_t m, typename I, typename E = void>
03763
               struct bernstein_helper {};
03764
03765
               template<typename I>
03766
               struct bernstein_helper<0, 0, I> {
03767
                  using type = typename polynomial<I>::one;
03768
03769
03770
               template<size_t i, size_t m, typename I>
               struct bernstein_helperi, m, I, std::enable_if_t<
(m > 0) && (i == 0) » {
03771
03772
03773
                private:
03774
                   using P = polynomial<I>;
                public:
03775
03776
                   using type = typename P::template mul_t<
03777
                            typename P::template sub_t<typename P::one, typename P::X>,
03778
                            typename bernstein_helper<i, m-1, I>::type>;
03779
03780
03781
               template<size_t i, size_t m, typename I>
03782
               struct bernstein_helper<i, m, I, std::enable_if_t<  (m > 0) \&\& (i == m) \  \  \{ 
03783
03784
                private:
03785
                   using P = polynomial<I>;
03786
                public:
03787
                  using type = typename P::template mul_t<
03788
                            typename P::X,
03789
                            typename bernstein helper<i-1, m-1, I>::type>;
03790
               };
03791
03792
               template<size_t i, size_t m, typename I>
               struct bernstein_helper<i, m, I, std::enable_if_t<  (m > 0) \&\& (i > 0) \&\& (i < m) * \{
03793
03794
03795
                private:
03796
                   using P = polynomial<I>;
03797
                public:
03798
                   using type = typename P::template add_t<
03799
                            typename P::template mul_t<
                                typename P::template sub_t<typename P::one, typename P::X>,
03800
                                typename bernstein_helper<i, m-1, I>::type>,
03801
03802
                            typename P::template mul_t<
03803
                                typename P::X,
03804
                                typename bernstein_helper<i-1, m-1, I>::type»;
03805
           } // namespace internal
03806
03807
03808
           // AllOne polynomials
03809
          namespace internal {
03810
               template<size_t deg, typename I>
03811
               struct AllOneHelper {
03812
                   using type = aerobus::add_t<
                        typename polynomial<I>::one,
03813
                        typename aerobus::mul_t<</pre>
03814
                            typename polynomial<I>::X,
03816
                            typename AllOneHelper<deg-1, I>::type
03817
03818
               };
03819
03820
               template<typename I>
03821
               struct AllOneHelper<0, I> {
03822
                  using type = typename polynomial<I>::one;
03823
03824
          } // namespace internal
03825
          // Bessel polynomials
03826
03827
          namespace internal {
03828
               template<size_t deg, typename I>
03829
               struct BesselHelper {
                private:
03830
                   using P = polynomial<I>;
using factor = typename P::template monomial_t<</pre>
03831
03832
                        typename I::template inject_constant_t<(2*deg - 1)>,
03833
03834
03835
                public:
03836
                   using type = typename P::template add_t<</pre>
03837
                        typename P::template mul_t<
03838
                            factor.
03839
                            typename BesselHelper<deg-1, I>::type
03840
03841
                        typename BesselHelper<deg-2, I>::type
03842
03843
               } ;
03844
03845
               template<tvpename I>
```

```
struct BesselHelper<0, I> {
                               using type = typename polynomial<I>::one;
03847
03848
                         };
03849
03850
                         template<typename I>
03851
                         struct BesselHelper<1, I> {
                         private:
03852
03853
                                using P = polynomial<I>;
                           public:
03854
03855
                               using type = typename P::template add_t<</pre>
03856
                                       typename P::one,
03857
                                       typename P::X
03858
03859
03860
                 } // namespace internal
03861
03862
                 namespace known_polynomials {
                        enum hermite_kind {
    probabilist,
03864
03866
03868
                                physicist
03869
03870
                 }
03871
                  // hermite
03872
03873
                 namespace internal {
03874
                       template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03875
                         struct hermite_helper {};
03876
03877
                         template<size_t deg, typename I>
                         \verb|struct hermite_helper<| deg, known_polynomials::hermite_kind::probabilist, I> \{ | learning | le
03878
03879
                          private:
03880
                               using hnm1 = typename hermite_helper<deg - 1,
          known_polynomials::hermite_kind::probabilist, I>::type;
03881
                               using hnm2 = typename hermite_helper<deg - 2,
          known_polynomials::hermite_kind::probabilist, I>::type;
03882
03883
                          public:
03884
                                using type = typename polynomial<I>::template sub_t<
03885
                                       typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnm1>,
03886
                                        typename polynomial<I>::template mul_t<
03887
                                               typename polynomial<I>::template inject_constant_t<deg - 1>,
03888
                                              hnm2
03889
03890
                                >;
03891
                        };
03892
03893
                         template<size_t deg, typename I>
03894
                         struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03895
                          private:
                                using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
03896
          I>::type;
03897
                                using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
          I>::type;
03898
03899
                           public:
03900
                                using type = typename polynomial<I>::template sub t<
                                        // 2X Hn-1
03901
03902
                                        typename polynomial<I>::template mul_t<
03903
                                               typename pi64::val<typename I::template inject_constant_t<2>,
03904
                                              typename I::zero>, hnm1>,
03905
03906
                                       typename polynomial<I>::template mul_t<</pre>
03907
                                               typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03908
03909
03910
                                >;
03911
                         };
03912
03913
                         template<tvpename I>
03914
                         struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03915
                               using type = typename polynomial<I>::one;
03916
03917
03918
                         template<typename I>
03919
                         struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03920
                                using type = typename polynomial<I>::X;
03921
03922
03923
                         template<typename I>
03924
                         struct hermite helper<0, known polynomials::hermite kind::physicist, I> {
03925
                                using type = typename pi64::one;
03926
03927
03928
                         template<typename I>
                         struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
    // 2X
03929
03930
03931
                                using type = typename polynomial<I>::template val<</pre>
```

```
typename I::template inject_constant_t<2>,
03933
                       typename I::zero>;
03934
          } // namespace internal
03935
03936
03937
          // legendre
03938
          namespace internal {
03939
              template<size_t n, typename I>
03940
              struct legendre_helper {
               private:
03941
                  using Q = FractionField<I>;
03942
                  using PQ = polynomial<Q>;
03943
03944
                   // 1/n constant
                   // (2n-1)/n X
03945
03946
                   using fact_left = typename PQ::template monomial_t<</pre>
03947
                       makefraction_t<I,</pre>
                           \label{typename I::template inject_constant_t<2*n-1>,} \\
03948
03949
                           typename I::template inject_constant_t<n>
03950
                  1>;
03951
03952
                   // (n-1) / n
03953
                   using fact_right = typename PQ::template val<
                       makefraction_t<I,
03954
                           typename I::template inject_constant_t<n-1>,
03955
03956
                           typename I::template inject_constant_t<n>>;
03957
03958
               public:
03959
                  using type = PQ::template sub_t<
03960
                           typename PQ::template mul_t<
03961
                               fact left.
03962
                               typename legendre_helper<n-1, I>::type
03963
03964
                           typename PQ::template mul_t<
03965
                               fact_right,
03966
                               typename legendre_helper<n-2, I>::type
03967
03968
                      >;
03969
              } ;
03970
03971
              template<typename I>
03972
              struct legendre_helper<0, I> {
                  using type = typename polynomial<FractionField<I>::one;
03973
03974
03975
03976
              template<typename I>
03977
              struct legendre_helper<1, I> {
03978
                  using type = typename polynomial<FractionField<I>::X;
03979
          } // namespace internal
03980
03981
03982
          // bernoulli polynomials
03983
          namespace internal {
03984
              template<size_t n>
03985
              struct bernoulli_coeff {
03986
                  template<typename T, size_t i>
03987
                  struct inner {
03988
                   private:
03989
                      using F = FractionField<T>;
03990
                    public:
03991
                      using type = typename F::template mul_t<</pre>
                           typename F::template inject_ring_t<combination_t<T, i, n»,
03992
                           bernoulli_t<T, n-i>
03993
03994
                      >;
03995
                  };
03996
03997
          } // namespace internal
03998
03999
          namespace internal {
04000
              template<size t n>
              struct touchard_coeff {
04001
04002
                  template<typename T, size_t i>
                  struct inner {
04003
04004
                       using type = stirling_2_t<T, n, i>;
04005
                  };
04006
04007
          } // namespace internal
04008
04009
          namespace internal {
04010
              template<typename I = aerobus::i64>
              struct AbelHelper {
04011
04012
               private:
04013
                  using P = aerobus::polynomial<I>;
04014
04015
               public:
04016
                  // to keep recursion working, we need to operate on a\!*\!n and not just a
04017
                  template<size_t deg, I::inner_type an>
04018
                  struct Inner {
```

```
04019
                         // abel(n, a) = (x-an) * abel(n-1, a)
                         using type = typename aerobus::mul_t<</pre>
04020
04021
                              typename Inner<deg-1, an>::type,
04022
                             typename aerobus::sub_t<typename P::X, typename P::template inject_constant_t<an>>
04023
04024
                    };
04025
04026
                    // abel(0, a) = 1
                    template<I::inner_type an>
04027
04028
                    struct Inner<0, an>
04029
                         using type = P::one;
04030
04031
04032
                    // abel(1, a) = X
04033
                    template<I::inner_type an>
                    struct Inner<1, an>
04034
04035
                         using type = P::X;
04036
                    };
04037
                };
04038
          } // namespace internal
04039
04041
           namespace known_polynomials {
04042
               template<size_t n, auto a, typename I = aerobus::i64>
using abel = typename internal::AbelHelper<I>::template Inner<n, a*n>::type;
04051
04052
04053
                template <size_t deg, typename I = aerobus::i64>
04061
04062
                using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
04063
04073
                template <size_t deg, typename I = aerobus::i64>
04074
               using chebyshev U = typename internal::chebyshev helper<2, deg, I>::type;
04075
04085
                template <size_t deg, typename I = aerobus::i64>
04086
                using laguerre = typename internal::laguerre_helper<deg, I>::type;
04087
                template <size_t deg, typename I = aerobus::i64>
04094
04095
               using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
      I>::type;
04096
04103
                template <size_t deg, typename I = aerobus::i64>
04104
                using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
04105
               template<size_t i, size_t m, typename I = aerobus::i64>
using bernstein = typename internal::bernstein_helper<i, m, I>::type;
04116
04117
04118
                template<size_t deg, typename I = aerobus::i64>
04128
04129
                using legendre = typename internal::legendre_helper<deg, I>::type;
04130
04140
                template<size_t deg, typename I = aerobus::i64>
04141
                using bernoulli = taylor<I, internal::bernoulli_coeff<deq>::template inner, deq>;
04142
04149
                template<size_t deg, typename I = aerobus::i64>
04150
                using allone = typename internal::AllOneHelper<deg, I>::type;
04151
                template<size_t deg, typename I = aerobus::i64>
04159
04160
               using bessel = typename internal::BesselHelper<deg, I>::type;
04161
04169
                template<size_t deg, typename I = aerobus::i64>
04170
               using touchard = taylor<I, internal::touchard_coeff<deg>::template inner, deg>;
04171
              // namespace known_polynomials
04172 } // namespace aerobus
04173
04174
04175 #ifdef AEROBUS_CONWAY_IMPORTS
04176
04177 // conway polynomials
04178 namespace aerobus {
          template<int p, int n>
04182
04183
           struct ConwayPolynomial {};
04185 #ifndef DO_NOT_DOCUMENT
        #define ZPZV ZPZ::template val
04186
          #define POLYV aerobus::polynomial<ZPZ>::template val
04187
           template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
04188
      POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
04189
           template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
04190
          template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
04191
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; // NOLINT
template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
04192
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT
04193
          template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>; }; // NOLINT template<> struct ConwayPolynomial<2, 7> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; ZPZV<1>; // NOLINT
04194
```

```
template<> struct ConwayPolynomial<2, 8> { using ZPZ = aerobus::zpz<2>; using type =
04195
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT
                                                         template<> struct ConwayPolynomial<2, 9> { using ZPZ = aerobus::zpz<2>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; //
                                     NOLINT
                                                           template<> struct ConwayPolynomial<2, 10> { using ZPZ = aerobus::zpz<2>; using type =
04197
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                                     ZPZV<1»; }; // NOLINT</pre>
                                                         template<> struct ConwayPolynomial<2, 11> { using ZPZ = aerobus::zpz<2>; using type =
04198
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
                                                          template<> struct ConwayPolynomial<2, 12> { using ZPZ = aerobus::zpz<2>; using type
04199
                                   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
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                     template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , Z
04201
                                     ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                                           template<> struct ConwayPolynomial<2, 15> { using ZPZ = aerobus::zpz<2>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT
   template<> struct ConwayPolynomial<2, 16> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04203
                                     ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                                         template<> struct ConwayPolynomial<2, 17> { using ZPZ = aerobus::zpz<2>; using type =
 04204
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     04205
                                   template<> struct ConwayPolynomial<2, 18> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1</pre>
                                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1); };</pre>
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       // NOLINT
                                                             template<> struct ConwayPolynomial<2, 19> { using ZPZ = aerobus::zpz<2>; using type
 04206
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
                                     NOLINT
                                     template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
04207
                                      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>; };
                                                          template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
                                   POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
                                                           template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
04209
                                   POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2»; }; // NOLINT
                                                              template<> struct ConwayPolynomial<3, 3> { using ZPZ = aerobus::zpz<3>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT
                                                         template<> struct ConwayPolynomial<3, 4> { using ZPZ = aerobus::zpz<3>; using type =
 04211
                                  POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; }; // NOLINT
 04212
                                                             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
04213
                                                             template<> struct ConwayPolynomial<3, 6> { using ZPZ = aerobus::zpz<3>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2
 04214
                                                           template<> struct ConwayPolynomial<3, 7> { using ZPZ = aerobus::zpz<3>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<1»; }; // NOLINT
04215
                                                          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
                                                              template<> struct ConwayPolynomial<3, 9> { using ZPZ = aerobus::zpz<3>; using type
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; //
04217
                                                              template<> struct ConwayPolynomial<3, 10> { using ZPZ = aerobus::zpz<3>; using type
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
                                     ZPZV<2»; }; // NOLINT</pre>
                                                             template<> struct ConwayPolynomial<3, 11> { using ZPZ = aerobus::zpz<3>; using type
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>,
                                      ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
04219
                                                        template<> struct ConwayPolynomial<3, 12> { using ZPZ = aerobus::zpz<3>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                                           template<> struct ConwayPolynomial<3, 13> { using ZPZ = aerobus::zpz<3>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT</pre>
                                                         template<> struct ConwayPolynomial<3, 14> { using ZPZ = aerobus::zpz<3>; using type =
04221
                                   POLYY<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<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<3>, ZPZV<3 , ZPZV<3
04222
                                     ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<3, 16> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
04223
04224
                                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT</pre>
 04225
                                                          template<> struct ConwayPolynomial<3, 18> { 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<1>, ZPZV<1
, ZPZV<1
                                   ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>; }; // NOLINT
template<> struct ConwayPolynomial<3, 19> { using ZPZ = aerobus::zpz<3>; using type =
04226
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; //</pre>
                               NOLINT
                              template<> struct ConwayPolynomial<3, 20> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>,
ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<
04227
                                                    template<> struct ConwayPolynomial<5, 1> { using ZPZ = aerobus::zpz<5>; using type :
 04228
                              POLYV<ZPZV<1>, ZPZV<3»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<5, 2> { using ZPZ = aerobus::zpz<5>; using type =
04229
                              POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<5, 3> { using ZPZ = aerobus::zpz<5>; using type =
 04230
                             POLYV<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<3»; ); // NOLINT template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
 04232
                                                  template<> struct ConwayPolynomial<5, 5> { using ZPZ = aerobus::zpz<5>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<3»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<5, 6> { using ZPZ = aerobus::zpz<5>; using type =
04233
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<5, 7> { using ZPZ = aerobus::zpz<5>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZV<3
 04235
                                                   template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
                              template<> struct ConwayPolynomial<5, 9> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
04236
                                                 template<> struct ConwayPolynomial<5, 10> { using ZPZ = aerobus::zpz<5>; using type =
04237
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<1>,
                               ZPZV<2»; }; // NOLINT</pre>
04238
                                                 template<> struct ConwayPolynomial<5, 11> { using ZPZ = aerobus::zpz<5>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<5, 12> { using ZPZ = aerobus::zpz<5>; using type =
 04239
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<4>, 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 =
04240
                               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<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<3>, ZPZV<5, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5, ZPZV<
                              template<> struct ConwayPolynomial<5, 15> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
04242
                              ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<5, ZPZV<3>; j; // NOLINT template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>,
                              ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<2>; }; // NOLINT
    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>, ZPZV<3 , ZPZ
                                                    template<> struct ConwayPolynomial<5, 18> { using ZPZ = aerobus::zpz<5>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1>, ZPZV<1 , ZPZV<1
                                ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0</pre>; // NOLINT
                              template<> struct ConwayPolynomial<5, 19> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
04246
                                ZPZV<0>, 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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3
                               ZPZV<4>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<0>, ZPZV<1>, ZPZV<2»; };</pre>
                                // NOLINT
                                                  template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
04248
                              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 =
 04250
                              POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4»; }; // NOLINT template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type =
04251
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
 04252
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT
 04253
                                                template<> struct ConwayPolynomial<7, 6> { using ZPZ = aerobus::zpz<7>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<4>, ZPZV<6>, ZPZV<6>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<7, 7> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<4»; }; // NOLINT
 04254
                                                    template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
                              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<6>, ZPZV<6>, ZPZV<3»; };
 04256
                                                 template<> struct ConwayPolynomial<7, 9> { using ZPZ = aerobus::zpz<7>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<4»; }; //
                              NOLINT
                              template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<3>,
04257
                               ZPZV<3»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<7, 11> { using ZPZ = aerobus::zpz<7>; using type
                              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<5>, ZPZV<0>, ZPZV<3»; }; // NOLINT</pre>
                                                            template<> struct ConwayPolynomial<7, 13> { using ZPZ = aerobus::zpz<7>; using type =
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      \text{ZPZV}<0>, \text{ZPZV}<6>, \text{ZPZV}<0>, \text{ZPZV}<4*; }; // NOLINT
                                     template<> struct ConwayPolynomial<7, 14> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , Z
                                       ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<3»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<7, 15> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>; ZPZV<0>; ZPZV<0>, ZPZV<0 , Z
04263
                                       ZPZV<3>, ZPZV<4>, ZPZV<1>, ZPZV<6>, ZPZV<2>, ZPZV<4>, ZPZV<3»; }; // NOLINT</pre>
                                                           template<> struct ConwayPolynomial<7, 17> { using ZPZ = aerobus::zpz<7>; using type
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial</pr>
7, 18> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<6>, ZPZV<1>,
04265
                                      ZPZV<6>, ZPZV<5>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<2>, ZPZV<2>, ZPZV<3»; };</pre>
                                     template<> struct ConwayPolynomial<7, 19> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
                                       \texttt{ZPZV} < \texttt{0} >, \ \texttt{Z
                                      NOLINT
                                                              template<> struct ConwayPolynomial<7, 20> { using ZPZ = aerobus::zpz<7>; using type
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<6>,
                                       ZPZV<2>, ZPZV<5>, ZPZV<2>, ZPZV<3>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<3>, ZPZV<3>
                                      // NOLINT
 04268
                                                               template<> struct ConwayPolynomial<11, 1> { using ZPZ = aerobus::zpz<11>; using type =
                                     POLYV<ZPZV<1>, ZPZV<9»; }; // NOLINT
                                                              template<> struct ConwayPolynomial<11, 2> { using ZPZ = aerobus::zpz<11>; using type =
04269
                                     POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2»; }; // NOLINT
                                                               template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<9»; ); // NOLINT
template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
 04271
                                    template<> struct ConwayPolynomial<11, 5> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<0>, ZPZV<9»; }; // NOLINT
04272
 04273
                                                                 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
 04274
                                                            template<> struct ConwayPolynomial<11, 7> { using ZPZ = aerobus::zpz<11>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<9»; }; // NOLINT
                                   template<> struct ConwayPolynomial<11, 8> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<2»; }; // NOLINT</pre>
 04275
                                                              template<> struct ConwayPolynomial<11, 9> { using ZPZ = aerobus::zpz<11>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<8>, ZPZV<9»; }; //
                                     template<> struct ConwayPolynomial<11, 10> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<10>, ZPZV<6>, ZPZV<6>,
                                      ZPZV<2»: }: // NOLINT</pre>
                                                               template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       ZPZV<10>, ZPZV<9»; }; // NOLINT</pre>
04279
                                                               template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type =
                                      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<2>, ZPZV<5>, ZPZV<5>,
                                      ZPZV<6>, ZPZV<5>, ZPZV<2»; }; // NOLINT</pre>
                                                                 template<> struct ConwayPolynomial<11, 13> { using ZPZ = aerobus::zpz<11>; using type
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<9>; }; // NOLINT
04281
                                                                 template<> struct ConwayPolynomial<11, 14> { using ZPZ = aerobus::zpz<11>; using type
                                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       ZPZV<7>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<0>, ZPZV<9»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<11, 16> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<3>, ZPZV<10>, ZPZV<10>, ZPZV<2»; }; // NOLINT
04283
                                                              template<> struct ConwayPolynomial<11, 17> { 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<4>, ZPZV<4>, ZPZV<9»; }; // NOLINT</pre>
04285
                                                           template<> struct ConwayPolynomial<11, 18> { using ZPZ = aerobus::zpz<11>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<2>; }; // NOLINT
                                     template<> struct ConwayPolynomial<11, 19> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04286
                                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<2>, ZPZV<9»; };</pre>
                                     template<> struct ConwayPolynomial<11, 20> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<5>, ZPZV<5</pre>
                                       // NOLINT
                                                               template<> struct ConwayPolynomial<13, 1> { using ZPZ = aerobus::zpz<13>; using type =
                                    POLYV<ZPZV<1>, ZPZV<11»; }; // NOLINT
                                                           template<> struct ConwayPolynomial<13, 2> { using ZPZ = aerobus::zpz<13>; using type =
 04289
                                   POLYV<ZPZV<1>, ZPZV<12>, ZPZV<2»; }; // NOLINT
                                                              template<> struct ConwayPolynomial<13, 3> { using ZPZ = aerobus::zpz<13>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11»; };
                                                                                                                                                                                                                      // NOLINT
                        template<> struct ConwayPolynomial<13, 4> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<13, 5> { using ZPZ = aerobus::zpz<13>; using type =
 04292
                         \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<4>, \verb"ZPZV<11"; \verb"}; "/ NOLINT" | NOLINT | N
04293
                                         template<> struct ConwayPolynomial<13, 6> { using ZPZ = aerobus::zpz<13>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1), ZPZV<11, ZPZV<2*; ; // NOLINT template<> struct ConwayPolynomial<13, 7> { using ZPZ = aerobus::zpz<13>; using type =
 04294
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<11»; }; // NOLINT
 04295
                                       template<> struct ConwayPolynomial<13, 8> { using ZPZ = aerobus::zpz<13>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<12>, ZPZV<2>, ZPZV<3>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<13, 9> { using ZPZ = aerobus::zpz<13>; using type =
04296
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<8>, ZPZV<12>, ZPZV<12>, ZPZV<11»; };
 04297
                                       template<> struct ConwayPolynomial<13, 10> { using ZPZ = aerobus::zpz<13>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>, ZPZV<2»; }; // NOLINT
                        template<> struct ConwayPolynomial<13, 11> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04298
                         ZPZV<3>, ZPZV<11»; };</pre>
                                                                                                                       // NOLINT
                        template<> struct ConwayPolynomial<13, 12> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<8>, ZPZV<81>, ZPZV<11>, ZPZV<3>, ZPZV<1>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<3>, ZPZV<1>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<1>, ZPZV<3>, ZPZV<3 , ZPZV<3
                        template<> struct ConwayPolynomial<13, 13> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04300
                                       template<> struct ConwayPolynomial<13, 14> { using ZPZ = aerobus::zpz<13>; using type =
04301
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<6>,
                        ZPZV<11>, ZPZV<7>, ZPZV<10>, ZPZV<10>, ZPZV<2»; }; // NOLINT
    template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3</pre>
04302
                        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 =
04303
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
                        ZPZV<8>, ZPZV<2>, ZPZV<12>, ZPZV<12>, ZPZV<6>, ZPZV<6>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<13, 17> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>; }; // NOLINT
04304
                                         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<1>, ZPZV<1>, ZPZV<4>, ZPZV<11>,
                        ZPZV<11>, ZPZV<9>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<13, 19> { using ZPZ = aerobus::zpz<13>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
04306
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<11»; }; //</pre>
04307
                                        template<> struct ConwayPolynomial<13, 20> { using ZPZ = aerobus::zpz<13>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<9>, ZPZV<9>, ZPZV<7>, ZPZV<8>, ZPZV<7>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<8>, ZPZV<8>, ZPZV<2»; };
                         // NOLINT
04308
                                         template<> struct ConwayPolynomial<17, 1> { using ZPZ = aerobus::zpz<17>; using type =
                        POLYV<ZPZV<1>, ZPZV<14»; }; // NOLINT
                                         template<> struct ConwayPolynomial<17, 2> { using ZPZ = aerobus::zpz<17>; using type =
                        POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»; }; // NOLINT
                                       template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
 04310
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
 04311
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<10>, ZPZV<3»; };
                                                                                                                                                                                                                                                          // NOLINT
                                      template<> struct ConwayPolynomial<17, 5> { using ZPZ = aerobus::zpz<17>; using type =
 04312
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT
 04313
                                          template<> struct ConwayPolynomial<17, 6> { using ZPZ = aerobus::zpz<17>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>; }; // NOLINT template<> struct ConwayPolynomial<17, 7> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<14*; }; // NOLINT
04314
                                          template<> struct ConwayPolynomial<17, 8> { using ZPZ = aerobus::zpz<17>; using type =
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<12>, ZPZV<0>, ZPZV<6>, ZPZV<3»; };
                        template<> struct ConwayPolynomial<17, 9> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<8>, ZPZV<14»; };</pre>
 04316
                          // NOLINT
04317
                                        template<> struct ConwayPolynomial<17, 10> { using ZPZ = aerobus::zpz<17>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<12>,
                          ZPZV<3»; }; // NOLINT</pre>
04318
                                      template<> struct ConwayPolynomial<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 =
04319
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<13>, ZPZV<6>,
                         ZPZV<14>, ZPZV<9>, ZPZV<3»; }; // NOLINT</pre>
04320
                                       template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14»; }; // NOLINT template<> struct ConwayPolynomial<17, 14> { using ZPZ = aerobus::zpz<17>; using type =
04321
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<8>,
                        ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::zpz<17>; using type =
 04322
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>,
                        ZPZV<4>, ZPZV<16>, ZPZV<6>, ZPZV<14>, ZPZV<14>, ZPZV<14»; }; // NOLINT
template<> struct ConwayPolynomial<17, 16> { using ZPZ = aerobus::zpz<17>; using type =
 04323
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<13>,
                             ZPZV<5>, ZPZV<2>, ZPZV<1>, ZPZV<13>, ZPZV<12>, ZPZV<1>, ZPZV<1>, ZPZV<3>; // NOLINT
template<> struct ConwayPolynomial<17, 17> { using ZPZ = aerobus::zpz<17>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<14»; }; // NOLINT
template<> struct ConwayPolynomial<17, 18> { using ZPZ = aerobus::zpz<17>; using type
04325
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<16>,
                              ZPZV<7>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<11>, ZPZV<13>, ZPZV<13>, ZPZV<9>, ZPZV<3»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<17, 19> { using ZPZ = aerobus::zpz<17>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 - , ZPZV<1 -
                             NOLINT
                             template<> struct ConwayPolynomial<17, 20> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>,
04327
                              ZPZV<16>, ZPZV<14>, ZPZV<13>, ZPZV<3>, ZPZV<14>, ZPZV<9>, ZPZV<1>, ZPZV<13>, ZPZV<2>, ZPZV<5>,
                              ZPZV<3»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<19, 1> { using ZPZ = aerobus::zpz<19>; using type =
                             POLYV<ZPZV<1>, ZPZV<17»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<19, 2> { using ZPZ = aerobus::zpz<19>; using type =
                             POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<19, 3> { using ZPZ = aerobus::zpz<19>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type =
04331
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 5> { using ZPZ = aerobus::zpz<19>; using type =
04332
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17»; }; // NOLINT
                                                template<> struct ConwayPolynomial<19, 6> { using ZPZ = aerobus::zpz<19>; using type =
04333
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<17>, ZPZV<16>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 7> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<17»; }; // NOLINT
04334
                                               template<> struct ConwayPolynomial<19, 8> { using ZPZ = aerobus::zpz<19>; using type =
04335
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<10>, ZPZV<3>, ZPZV<2»; };
                                                 template<> struct ConwayPolynomial<19, 9> { using ZPZ = aerobus::zpz<19>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<16>, ZPZV<17»; };
                             // NOLINT
                                               template<> struct ConwayPolynomial<19, 10> { using ZPZ = aerobus::zpz<19>; using type
04337
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<18>, ZPZV<13>, ZPZV<17>, ZPZV<3>, ZPZV<4>,
                             ZPZV<2»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<19, 11> { using ZPZ = aerobus::zpz<19>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04339
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<17»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<19, 14> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<11>,
                             ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<7>, ZPZV<2), {PZV<7}, ZPZV<2); }; // NOLINT template<> struct ConwayPolynomial<19, 15> { using ZPZ = aerobus::zpz<19>; using type
04342
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<10>, ZPZV<1>, ZPZV<10>, ZPZV<1>, ZPZV<10>, ZPZV
                              ZPZV<11>, ZPZV<13>, ZPZV<15>, ZPZV<14>, ZPZV<0>, ZPZV<17»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<19, 16> { using ZPZ = aerobus::zpz<19>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04343
                             ZPZV<13>, ZPZV<0>, ZPZV<15>, ZPZV<0>, ZPZV<5>, ZPZV<2>, ZPZV<15>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<19, 17> { using ZPZ = aerobus::zpz<19>; using type
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<17»; };</pre>
04345
                                                  template<> struct ConwayPolynomial<19, 18> { using ZPZ = aerobus::zpz<19>; using type
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<2», ZPZV<1>, ZPZV<3>, ZPZV<3 , ZPZV<3
04346
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<117»; };</pre>
                             NOLINT
                             template<> struct ConwayPolynomial<19, 20> { using ZPZ = aerobus::zpz<19>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>,
ZPZV<13>, ZPZV<4>, ZPZV<4>, ZPZV<7>, ZPZV<8>, ZPZV<6>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<6>, ZPZV<11>, ZPZV<2»;</pre>
                             }; // NOLINT
                                                 template<> struct ConwayPolynomial<23, 1> { using ZPZ = aerobus::zpz<23>; using type =
                             POLYV<ZPZV<1>, ZPZV<18»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<23, 2> { using ZPZ = aerobus::zpz<23>; using type =
04349
                             POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
04350
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; }; // NOLINT
                                               template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
04351
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type =
04352
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<18»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<23, 6> { using ZPZ = aerobus::zpz<23>; using type =
04353
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<9>, ZPZV<1>, ZPZV<5»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<23, 7> { using ZPZ = aerobus::zpz<23>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<18»; };
04355
                                             template<> struct ConwayPolynomial<23, 8> { using ZPZ = aerobus::zpz<23>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<20>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<23, 9> { using ZPZ = aerobus::zpz<23>; using type =
04356
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<8>, ZPZV<9>, ZPZV<18»; };
 04357
                                                             template<> struct ConwayPolynomial<23, 10> { using ZPZ = aerobus::zpz<23>; using type =
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6 , ZPZV<6
                                       ZPZV<5»; }; // NOLINT</pre>
                                                                template<> struct ConwayPolynomial<23, 11> { using ZPZ = aerobus::zpz<23>; using type =
 04358
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<7>, ZPZV<18»; }; // NOLINT</pre>
                                                             template<> struct ConwayPolynomial<23, 12> { using ZPZ = aerobus::zpz<23>; using type
 04359
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<21>, ZPZV<21>, ZPZV<15>, ZPZV<14>, ZPZV<12>,
                                       ZPZV<18>, ZPZV<12>, ZPZV<5»; }; // NOLINT</pre>
                                                             template<> struct ConwayPolynomial<23, 13> { using ZPZ = aerobus::zpz<23>; using type =
 04360
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<18»; };</pre>
                                                                                                                                                                                                                                                                                                          // NOLINT
                                                            template<> struct ConwayPolynomial<23, 14> { using ZPZ = aerobus::zpz<23>; using type =
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<1>, ZPZV<1>, ZPZV<18>, ZPZV<19>, ZPZV<1>, ZPZV<22>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<23, 15> { using ZPZ = aerobus::zpz<23>; using type =
 04362
                                       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
                                      ZPZV<19>, ZPZV<16>, ZPZV<13>, ZPZV<1>, ZPZV<14>, ZPZV<17>, ZPZV<5»; }; // NOLINT
    template<> struct ConwayPolynomial<23, 17> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0</pre>
 04364
                                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<18»; }; // NOLINT</pre>
                                                            template<> struct ConwayPolynomial<23, 18> { using ZPZ = aerobus::zpz<23>; using type =
 04365
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1
, ZPZV<1
                                      ZPZV<18>, ZPZV<3>, ZPZV<16>, ZPZV<21>, ZPZV<0>, ZPZV<11>, ZPZV<3>, ZPZV<19, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<23, 19> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04366
                                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5</pre>
                                                              template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type =
 04367
                                      POLYV<ZPZV<1>, ZPZV<27»; }; // NOLINT
                                                                template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
 04368
                                      POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; }; // NOLINT
                                                                template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<27»; }; // NOLINT
  04370
                                                             template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
                                    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<0>, ZPZV<3>, ZPZV<27»; }; // NOLINT
 04371
                                                               template<> struct ConwayPolynomial<29, 6> { using ZPZ = aerobus::zpz<29>; using type =
  04372
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<13>, ZPZV<23>, ZPZV<29; }; // NOLINT template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type =
  04373
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<27»; }; // NOLINT
  04374
                                                            template<> struct ConwayPolynomial<29, 8> { using ZPZ = aerobus::zpz<29>; using type =
                                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<2»; };
                                      NOLINT
 04375
                                                               template<> struct ConwayPolynomial<29, 9> { using ZPZ = aerobus::zpz<29>; using type
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<22>, ZPZV<22>, ZPZV<27»; };
                                         // NOLINT
 04376
                                                                template<> struct ConwayPolynomial<29, 10> { using ZPZ = aerobus::zpz<29>; using type =
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<8>, ZPZV<17>, ZPZV<2>, ZPZV<2>,
                                       ZPZV<2»; }; // NOLINT</pre>
                                                                  template<> struct ConwayPolynomial<29, 11> { using ZPZ = aerobus::zpz<29>; using type
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       ZPZV<8>, ZPZV<27»; }; // NOLINT</pre>
 04378
                                                                  template<> struct ConwayPolynomial<29, 12> { using ZPZ = aerobus::zpz<29>; using type
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<28>, ZPZV<9>, ZPZV<16>, ZPZV<25>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2); }; // NOLINT
                                                                template<> struct ConwayPolynomial<29, 13> { using ZPZ = aerobus::zpz<29>; using type
                                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                            template<> struct ConwayPolynomial<29, 14> { using ZPZ = aerobus::zpz<29>; using type =
 04380
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<1>, ZPZV<3>, ZPZV<1>, ZPZV<3>, ZPZV<1>, ZPZV<10>, ZPZV<
                                       ZPZV<21>, ZPZV<18>, ZPZV<27>, ZPZV<5>, ZPZV<2»; }; // NOLINT</pre>
                                                              template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::zpz<29>; using type =
 04381
                                       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 =
 04382
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; using type =
 04383
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        \texttt{ZPZV} < \texttt{O} >, \ \texttt{ZPZV} < \texttt{ZPZV} < \texttt{O} >, \ \texttt{ZPZV} < \texttt{Z
                                      template<> struct ConwayPolynomial<29, 18> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<1>, ZPZV<1>,
 04384
                                      ZPZV<6>, ZPZV<26>, ZPZV<2>, ZPZV<10>, ZPZV<10>, ZPZV<16>, ZPZV<19>, ZPZV<14>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<29, 19> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 
 04385
                                         ZPZV<0>, ZPZV<0>
                                       NOLINT
  04386
                                                              template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
                                     POLYV<ZPZV<1>, ZPZV<28»; }; // NOLINT
                                                              template<> struct ConwayPolynomial<31, 2> { using ZPZ = aerobus::zpz<31>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<29>, ZPZV<3»; };
                                                template<> struct ConwayPolynomial<31, 3> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; // NOLINT template<> struct ConwayPolynomial<31, 4> { using ZPZ = aerobus::zpz<31>; using type =
 04389
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 5> { using ZPZ = aerobus::zpz<31>; using type =
04390
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; }; // NOLINT
                                                template<> struct ConwayPolynomial<31, 6> { using ZPZ = aerobus::zpz<31>; using type =
 04391
                            template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<28»; }; // NOLINT
template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type =
 04392
04393
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<12>, ZPZV<24>, ZPZV<24>, ZPZV<3»; }; //
 04394
                                            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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<26>, ZPZV<26>, ZPZV<13>, ZPZV<13
04395
                             ZPZV<3»: }: // NOLINT
                            template<> struct ConwayPolynomial<31, 11> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                             ZPZV<20>, ZPZV<28»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<31, 12> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<14>, ZPZV<28>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<25>, ZPZV<12>, ZPZV<3»; }; // NOLINT
04397
                                            template<> struct ConwayPolynomial<31, 13> { using ZPZ = aerobus::zpz<31>; using type =
04398
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<28»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<31, 14> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<10, ZPZV<5>, ZPZV<1>, ZPZV<1
04399
                             ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<6>, ZPZV<3»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type =
04400
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<29>, ZPZV<12>, ZPZV<13>, ZPZV<23>, ZPZV<25>, ZPZV<28*; }; // NOLINT
    template<> struct ConwayPolynomial<31, 16> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
04401
                             ZPZV<24>, ZPZV<26>, ZPZV<28>, ZPZV<11>, ZPZV<19>, ZPZV<27>, ZPZV<3»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<31, 17> { using ZPZ = aerobus::zpz<31>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             template<> struct ConwayPolynomial<31, 18> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<27>, ZPZV<27>, ZPZV<24>, ZPZV<24>, ZPZV<25>, ZPZV<25>, ZPZV<3>, ZPZV<3»; }; // NOLINT
04403
                                               template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                                                                                        ZPZV<0>,
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0</pre>
                             NOLINT
04405
                                               template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
                            POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                               template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
                            POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2»; }; // NOLINT
 04407
                                                template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
 04408
                           POLYVCZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<37, 5> { using ZPZ = aerobus::zpz<37>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
 04410
                            template<> struct ConwayPolynomial<37, 7> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<35»; }; // NOLINT template<> struct ConwayPolynomial<37, 8> { using ZPZ = aerobus::zpz<37>; using type =
 04411
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<20>, ZPZV<27>, ZPZV<1>, ZPZV<2»; }; // NOLINT
04413
                                             template<> struct ConwayPolynomial<37, 9> { using ZPZ = aerobus::zpz<37>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<32>, ZPZV<32>, ZPZV<35»; };
                             // NOLINT
04414
                                               template<> struct ConwayPolynomial<37, 10> { using ZPZ = aerobus::zpz<37>; using type =
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<29>, ZPZV<18>, ZPZV<18>, ZPZV<11>, ZPZV<4>,
                             ZPZV<2»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<37, 11> { using ZPZ = aerobus::zpz<37>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<2>, ZPZV<35»; }; // NOLINT</pre>
04416
                                                template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<31>, ZPZV<10>, ZPZV<23>, ZPZV<23>, ZPZV<18>, ZPZV<33>, ZPZV<28; }; // NOLINT
                                             template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT</pre>
                            ZPZV<U>, ZPZV<U>, ZPZV<O>, ZPZV<O , ZPZV<O , ZPZV<O>, ZPZV<O , ZPZV<O 
04418
                             ZPZV<32>, ZPZV<16>, ZPZV<1>, ZPZV<9>, ZPZV<2»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<37, 15> { using ZPZ = aerobus::zpz<37>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<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>,
 04420
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35»; };</pre>
                   template<> struct ConwayPolynomial<37, 18> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<20>, ZPZV<1>, ZPZV<20>, ZPZV<1>, ZPZV<10>, ZPZV<20>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<0>, ZPZV<0 , ZPZV
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<36>, ZPZV<35»; }; //</pre>
                                 template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type =
04423
                   POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                 template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
04424
                   POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6»; }; // NOLINT
04425
                                 template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<35»; }; // NOLINT
04426
                               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 =
04427
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40, ZPZV<14, ZPZV<15; }; // NOLINT template<> struct ConwayPolynomial<41, 6> { using ZPZ = aerobus::zpz<41>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<33>, ZPZV<39>, ZPZV<6>, ZPZV<6»; }; // NOLINT
                                  template<> struct ConwayPolynomial<41, 7> { using ZPZ = aerobus::zpz<41>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<35»; };
04430
                               template<> struct ConwayPolynomial<41, 8> { using ZPZ = aerobus::zpz<41>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<32>, ZPZV<32>, ZPZV<32>, ZPZV<32>, ZPZV<42); template<> struct ConwayPolynomial<41, 9> { using ZPZ = aerobus::zpz<41>; using type =
04431
                    POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<35>, ZPZV<35»; };
                   template<> struct ConwayPolynomial<41, 10> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<8>, ZPZV<20>, ZPZV<30>,
                    ZPZV<6»; }; // NOLINT
                   template<> struct ConwayPolynomial<41, 11> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                     ZPZV<20>, ZPZV<35»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<41, 12> { using ZPZ = aerobus::zpz<41>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26>, ZPZV<13>, ZPZV<34>, ZPZV<24>, ZPZV<21>, ZPZV<27>, ZPZV<6»; }; // NOLINT
                                 template<> struct ConwayPolynomial<41, 13> { using ZPZ = aerobus::zpz<41>; using type =
04435
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                    ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<35»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<41, 14> { using ZPZ = aerobus::zpz<41>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<1>, ZPZV<15>, ZPZV<4>,
                   ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6>; }; // NOLINT
    template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
04437
                   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
                    template<> struct ConwayPolynomial<41, 18> { using ZPZ = aerobus::zpz<41>; using type
04439
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<7>, ZPZV<20>,
                   ZPZV<23>, ZPZV<35>, ZPZV<38>, ZPZV<24>, ZPZV<12>, ZPZV<29>, ZPZV<10>, ZPZV<6>, ZPZV<6>, ZPZV<6»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35»; }; //</pre>
                    NOLINT
04441
                                 template<> struct ConwayPolynomial<43, 1> { using ZPZ = aerobus::zpz<43>; using type =
                    POLYV<ZPZV<1>, ZPZV<40»; }; // NOLINT
                               template<> struct ConwayPolynomial<43, 2> { using ZPZ = aerobus::zpz<43>; using type =
                   POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3»; }; // NOLINT
04443
                                  template<> struct ConwayPolynomial<43, 3> { using ZPZ = aerobus::zpz<43>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40»; }; // NOLINT
template<> struct ConwayPolynomial<43, 4> { using ZPZ = aerobus::zpz<43>; using type =
04444
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3»; };
                                                                                                                                                                                                             // NOLINT
                                  template<> struct ConwayPolynomial<43, 5> { using ZPZ = aerobus::zpz<43>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<40»; }; // NOLINT
04446
                                template<> struct ConwayPolynomial<43, 6> { using ZPZ = aerobus::zpz<43>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<21>, ZPZV<3»; }; // NOLINT
04447
                                template<> struct ConwayPolynomial<43, 7> { using ZPZ = aerobus::zpz<43>; using type =
                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<7>, ZPZV<7>, ZPZV<40»; }; // NOLINT
                                  template<> struct ConwayPolynomial<43, 8> { using ZPZ = aerobus::zpz<43>, using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<39>, ZPZV<20>, ZPZV<24>, ZPZV<3»; };
                   template<> struct ConwayPolynomial<43, 9> { using ZPZ = aerobus::zpz<43>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<40»; };</pre>
                     // NOLINT
                                  template<> struct ConwayPolynomial<43, 10> { using ZPZ = aerobus::zpz<43>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<26>, ZPZV<36>, ZPZV<36>, ZPZV<5>, ZPZV<27>, ZPZV<24>,
                     ZPZV<3»; }; // NOLINT</pre>
                   template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, 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>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38>, ZPZV<38
                               template<> struct ConwayPolynomial<43, 13> { using ZPZ = aerobus::zpz<43>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
```

```
template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<38>, ZPZV<22>, ZPZV<24>,
                                     ZPZV<37>, ZPZV<18>, ZPZV<4>, ZPZV<19>, ZPZV<3»; }; // NOLINT</pre>
                                                         template<> struct ConwayPolynomial<43, 15> { using ZPZ = aerobus::zpz<43>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZV<3
                                    ZPZV<22>, ZPZV<42>, ZPZV<45, ZPZV<45>, ZPZV<45>, ZPZV<47>, ZPZV<40»; }; // NOLINT template<> struct ConwayPolynomial<43, 17> { using ZPZ = aerobus::zpz<43>; using type
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<40»; }; // NOLINT</pre>
                                    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<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3+, ZPZV<3+,
 04457
 04458
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<0>
04459
                                                             template<> struct ConwayPolynomial<47, 1> { using ZPZ = aerobus::zpz<47>; using type =
                                    POLYV<ZPZV<1>, ZPZV<42»; }; // NOLINT
                                                              template<> struct ConwayPolynomial<47, 2> { using ZPZ = aerobus::zpz<47>; using type =
                                    POLYV<ZPZV<1>, ZPZV<45>, ZPZV<5»; }; // NOLINT
                                                               template<> struct ConwayPolynomial<47, 3> { using ZPZ = aerobus::zpz<47>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42»; }; // NOLINT template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type =
 04462
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<47, 5> { using ZPZ = aerobus::zpz<47>; using type =
  04463
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT
                                                            template<> struct ConwayPolynomial<47, 6> { using ZPZ = aerobus::zpz<47>; using type =
  04464
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<35>, ZPZV<9>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<47, 7> { using ZPZ = aerobus::zpz<47>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
  04465
 04466
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<19>, ZPZV<19>, ZPZV<3>, ZPZV<5»; };
                                                             template<> struct ConwayPolynomial<47, 9> { using ZPZ = aerobus::zpz<47>; using type
  04467
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<42»; };
                                     // NOLINT
                                                          template<> struct ConwayPolynomial<47, 10> { using ZPZ = aerobus::zpz<47>; using type =
 04468
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42>, ZPZV<14>, ZPZV<18>, ZPZV<45>, ZPZV<45>,
                                     ZPZV<5»; }; // NOLINT</pre>
                                                             template<> struct ConwayPolynomial<47, 11> { using ZPZ = aerobus::zpz<47>; using type
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     ZPZV<6>, ZPZV<42»; }; // NOLINT
                                   template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<46>, ZPZV<40>, ZPZV<35>, ZPZV<12>, ZPZV<46>, ZPZV<46>, ZPZV<40>, ZPZV<35>; }; // NOLINT
 04470
                                                              template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<42»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<47, 14> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<30>, ZPZV<30>
                                    ZPZV<17>, ZPZV<24>, ZPZV<9>, ZPZV<32>, ZPZV<5»; ); // NOLINT template<> struct ConwayPolynomial<47, 15> { using ZPZ = aerobus::zpz<47>; using type
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<31>, ZPZV<14>, ZPZV<42>, ZPZV<13>, ZPZV<17>, ZPZV<42»; }; // NOLINT</pre>
                                    \label{eq:convayPolynomial} $$ \text{template} <> \text{struct ConwayPolynomial} <47, 17> $$ using $$ \text{ZPZ} = aerobus::zpz<47>$$; using type = $$ POLYV<2PZV<1>, $$ ZPZV<0>, $$ ZPZ
 04474
                                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<42»; }; // NOLINT
template<> struct ConwayPolynomial<47, 18> { using ZPZ = aerobus::zpz<47>; using type
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<41>, ZPZV<42>,
                                     ZPZV<26>, ZPZV<44>, ZPZV<22>, ZPZV<11>, ZPZV<5>, ZPZV<45>, ZPZV<33>, ZPZV<5»; }; // NOLINT</pre>
                                    \label{eq:convayPolynomial} $$ \text{template} > \text{struct ConwayPolynomial} < 47, 19> $ \{ \text{using ZPZ} = \text{aerobus}:: zpz < 47>; \text{using type} = POLYV < 2PZV < 1>, ZPZV < 0>, ZPZ
 04476
                                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42»; }; //</pre>
                                     NOLINT
                                                              template<> struct ConwayPolynomial<53, 1> { using ZPZ = aerobus::zpz<53>; using type =
                                   POLYV<ZPZV<1>, ZPZV<51»; }; // NOLINT
  04478
                                                          template<> struct ConwayPolynomial<53, 2> { using ZPZ = aerobus::zpz<53>; using type =
                                    POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2»; }; // NOLINT
                                                          template<> struct ConwayPolynomial533, 3> { using ZPZ = aerobus::zpz<53>; using type =
 04479
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT
                                                              template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
  04480
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<38>, ZPZV<2»; }; // NOLINT
                                                         template<> struct ConwayPolynomial<53, 5> { using ZPZ = aerobus::zpz<53>; using type =
  04481
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT template<> struct ConwayPolynomial<53, 6> { using ZPZ = aerobus::zpz<53>; using type =
  04482
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<4>, ZPZV<45>, ZPZV<45>, ZPZV<45>, ZPZV<53>; ); // NOLINT template<> struct ConwayPolynomial<53, 7> { using ZPZ = aerobus::zpz<53>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51»; };
                                                          template<> struct ConwayPolynomial<53, 8> { using ZPZ = aerobus::zpz<53>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<1>, ZPZV<1>, ZPZV<22; }; // NOLI template<> struct ConwayPolynomial<53, 9> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<51»; };
  04485
                                     // NOLINT
                                                          template<> struct ConwayPolynomial<53, 10> { using ZPZ = aerobus::zpz<53>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<27>, ZPZV<27>, ZPZV<29>,
                                    ZPZV<2»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<53, 11> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
  04487
```

```
ZPZV<15>, ZPZV<51»; };</pre>
                      template<> struct ConwayPolynomial<53, 12> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<34>, ZPZV<41>, ZPZV<13>, ZPZV<10>, ZPZV<42>,
                       ZPZV<34>, ZPZV<41>, ZPZV<2»; }; // NOLINT
                                      template<> struct ConwayPolynomial<53, 13> { using ZPZ = aerobus::zpz<53>; using type
04489
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<45>, ZPZV<45>, ZPZV<23>, ZPZV<52>,
                      ZPZV<0>, ZPZV<37>, ZPZV<12>, ZPZV<23>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<53, 15> { using ZPZ = aerobus::zpz<53>; using type =
04491
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>,
                      ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51»; }; // NOLINT
template<> struct ConwayPolynomial<53, 17> { using ZPZ = aerobus::zpz<53>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       template<> struct ConwayPolynomial<53, 18> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<52>, ZPZV<52>, ZPZV<52>, ZPZV<51>, ZPZV<27>, ZPZV<0>, ZPZV<39>, ZPZV<44>, ZPZV<6>, ZPZV<8>, ZPZV<16>, ZPZV<11>, ZPZV<2»; }; // NOLINT
                                     template<> struct ConwayPolynomial<53, 19> { using ZPZ = aerobus::zpz<53>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<51»; }; //</pre>
                       NOLINT
                                     template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
                      POLYV<ZPZV<1>, ZPZV<57»; }; // NOLINT
                                     template<> struct ConwayPolynomial<59, 2> { using ZPZ = aerobus::zpz<59>; using type =
                      POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2»; }; // NOLINT
04497
                                    template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; }; // NOLINT template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
04498
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<40>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<59, 5> { using ZPZ = aerobus::zpz<59>; using type =
04499
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<57»; }; // NOLINT
04500
                                   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<57»; }; // NOLINT
04501
                                     template<> struct ConwayPolynomial<59, 8> { using ZPZ = aerobus::zpz<59>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<32>, ZPZV<2>, ZPZV<50>, ZPZV<2»; };
04503
                                   template<> struct ConwayPolynomial<59, 9> { using ZPZ = aerobus::zpz<59>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<47>, ZPZV<57»; };
                       // NOLINT
04504
                                      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<49, ZPZV<39, ZPZV<15>,
                       ZPZV<2»; }; // NOLINT</pre>
04505
                                    template<> struct ConwayPolynomial<59, 11> { using ZPZ = aerobus::zpz<59>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type =
04506
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<21>, ZPZV<38>, ZPZV<8>, ZPZV<1>, ZPZV<2), ZPZV<38>, ZPZV<8>, ZPZV<1>, ZPZV<2»; }; // NOLINT
04507
                                    template<> struct ConwayPolynomial<59, 13> { using ZPZ = aerobus::zpz<59>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                // NOLINT
04508
                                      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</pre>
                      template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                      ZPZV<24>, ZPZV<23>, ZPZV<13>, ZPZV<39>, ZPZV<58>, ZPZV<57»; }; // NOLINT
    template<> struct ConwayPolynomial<59, 17> { using ZPZ = aerobus::zpz<59>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04510
                       template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59>; using type =
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<2>,
                      ZPZV<11>, ZPZV<14>, ZPZV<7>, ZPZV<44>, ZPZV<16>, ZPZV<47>, ZPZV<34>, ZPZV<32>, ZPZV<32>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<59, 19> { using ZPZ = aerobus::zpz<59>; using type =
04512
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<11>, ZPZV<57»; );</pre>
                                    template<> struct ConwayPolynomial<61, 1> { using ZPZ = aerobus::zpz<61>; using type =
                      POLYV<ZPZV<1>, ZPZV<59»; }; // NOLINT
                                      template<> struct ConwayPolynomial<61, 2> { using ZPZ = aerobus::zpz<61>; using type =
04514
                      POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; }; // NOLINT
                                      template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; }; // NOLINT template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
04516
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<61, 5> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59»; }; // NOLINT
04517
                                      template<> struct ConwayPolynomial<61, 6> { using ZPZ = aerobus::zpz<61>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<29>, ZPZV<29>, ZPZV<29>, ZPZV<29>; // NOLINT template<> struct ConwayPolynomial<61, 7> { using ZPZ = aerobus::zpz<61>; using type =
04519
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<59»; }; // NOLINT
04520
                      template<> struct ConwayPolynomial<61, 8> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<56>, ZPZV<56>, ZPZV<2»; }; // NOLINT</pre>
```

```
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<5>, ZPZV<50>, ZPZV<58>, ZPZV<59»; };
                                // NOLINT
                               template<> struct ConwayPolynomial<61, 10> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<15>, ZPZV<44>, ZPZV<16>, ZPZV<6>,
                                ZPZV<2»: }: // NOLINT
                                                    template<> struct ConwayPolynomial<61, 11> { using ZPZ = aerobus::zpz<61>; using type
                                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<18>, ZPZV<59»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<61, 12> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<42>, ZPZV<33>, ZPZV<8>, ZPZV<38>, ZPZV<14>,
ZPZV<15>, ZPZV<2s, }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type
04525
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<59»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<61, 14> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<48>, ZPZV<48>, ZPZV<26>, ZPZV<11>,
04526
                               ZPZV<8>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<48>, ZPZV<20; LFZVVI, ZFZVVI, ZFZV
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<35>, ZPZV<44>, ZPZV<25>, ZPZV<23>, ZPZV<51>, ZPZV<59»; }; // NOLINT</pre>
04528
                                                     template<> struct ConwayPolynomial<61, 17> { using ZPZ = aerobus::zpz<61>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 \texttt{ZPZV} < \texttt{0>, ZPZV} < \texttt{0.2.} < \texttt{0.2
                                                    template<> struct ConwayPolynomial<61, 18> { using ZPZ = aerobus::zpz<61>; using type
                                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<35>, ZPZV<36>, ZPZV<13>,
                                ZPZV<36>, ZPZV<4>, ZPZV<32>, ZPZV<57>, ZPZV<42>, ZPZV<25>, ZPZV<25>, ZPZV<52>, ZPZV<52»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<61, 19> { using ZPZ = aerobus::zpz<61>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; }; //</pre>
                                NOLINT
04531
                                                     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 =
                               POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
 04533
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<54>, ZPZV<2»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<67, 5> { using ZPZ = aerobus::zpz<67>; using type =
                             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 =
04536
                              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<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<65»; }; //
 04538
                                                   template<> struct ConwayPolynomial<67, 8> { using ZPZ = aerobus::zpz<67>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<46>, ZPZV<17>, ZPZV<64>, ZPZV<2»; }; //
                               template<> struct ConwayPolynomial<67, 9> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<55>, ZPZV<55>; ZPZV<65»; };
04539
                                 // NOLINT
                                                    template<> struct ConwayPolynomial<67, 10> { using ZPZ = aerobus::zpz<67>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<0>, ZPZV<16>, ZPZV<15, ZPZV<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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<65»; }; // NOLINT
04541
                                                template<> struct ConwayPolynomial<67, 12> { using ZPZ = aerobus::zpz<67>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57>, ZPZV<27>, ZPZV<4>, ZPZV<55>, ZPZV<64>, ZPZV<64>, ZPZV<21>, ZPZV<27>, ZPZV<22>, ZPZV<28; }; // NOLINT
                                                  template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type =
04543
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<5>; }; // NOLINT template<> struct ConwayPolynomial<67, 14> { using ZPZ = aerobus::zpz<67>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<17>, ZPZV<22>, ZPZV<5>,
                                ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<2»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<52>, ZPZV<41>, ZPZV<20>, ZPZV<21>, ZPZV<46>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<67, 17> { using ZPZ = aerobus::zpz<67>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<55>, ZPZV<65»; }; // NOLINT
    template<> struct ConwayPolynomial<67, 18> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<63>, ZPZV<5>, ZPZV<18>,
                               ZPZV<33>, ZPZV<55>, ZPZV<28>, ZPZV<29>, ZPZV<51>, ZPZV<60, ZPZV<59>, ZPZV<13>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<67, 19> { using ZPZ = aerobus::zpz<67>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<65»; }; //</pre>
                                NOLINT
04549
                                                  template<> struct ConwayPolynomial<71, 1> { using ZPZ = aerobus::zpz<71>; using type =
                              POLYV<ZPZV<1>, ZPZV<64»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<71, 2> { using ZPZ = aerobus::zpz<71>; using type =
                              POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7»; }; // NOLINT
 04551
                                                 template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; }; // NOLINT
template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, ZPZV<7»; }; // NOLINT
 04552
```

```
template<> struct ConwayPolynomial<71, 5> { using ZPZ = aerobus::zpz<71>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<71, 6> { using ZPZ = aerobus::zpz<71>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<13>, ZPZV<29>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<71, 7> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1, zpzv<0>; ZPZV<1>; using type = aerobus::zpz<71>; using type = demplate<> struct ConwayPolynomial<71, 8> { using ZPZ = aerobus::zpz<71>; using type =
 04555
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<22>, ZPZV<19>, ZPZV<7»; };
                               template<> struct ConwayPolynomial<71, 9> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43>, ZPZV<43>, ZPZV<62>, ZPZV<64»; };</pre>
04557
                                // NOLINT
                                                     template<> struct ConwayPolynomial<71, 10> { using ZPZ = aerobus::zpz<71>; using type =
04558
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<17>, ZPZV<26>, ZPZV<15, ZPZV<40>,
                                ZPZV<7»; }; // NOLINT</pre>
04559
                                                   template<> struct ConwayPolynomial<71, 11> { using ZPZ = aerobus::zpz<71>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<48>, ZPZV<64»; }; // NOLINT</pre>
                                                      template<> struct ConwayPolynomial<71, 12> { using ZPZ = aerobus::zpz<71>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<28>, ZPZV<29>, ZPZV<55>, ZPZV<21>,
                                ZPZV<58>, ZPZV<23>, ZPZV<7»; }; // NOLINT</pre>
04561
                                                     template<> struct ConwayPolynomial<71, 13> { using ZPZ = aerobus::zpz<71>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04562
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<32>, ZPZV<18>, ZPZV<52>, ZPZV<67>, ZPZV<49>, ZPZV<64»; }; // NOLINT</pre>
                                                  template<> struct ConwayPolynomial<71, 17> { using ZPZ = aerobus::zpz<71>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<64»; }; // NOLINT template<> struct ConwayPolynomial<71, 19> { using ZPZ = aerobus::zpz<71>; using type :
                                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<44»; }; //</pre>
04565
                                                     template<> struct ConwayPolynomial<73, 1> { using ZPZ = aerobus::zpz<73>; using type =
                               POLYV<ZPZV<1>, ZPZV<68»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<73, 2> { using ZPZ = aerobus::zpz<73>; using type =
04566
                               POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5»; }; // NOLINT
 04567
                                                      template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<68»; }; // NOLINT template<> struct ConwayPolynomial<73, 4> { using ZPZ = aerobus::zpz<73>; using type =
 04568
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<73, 5> { using ZPZ = aerobus::zpz<73>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<68»; }; // NOLINT
 04569
                                                      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
 04571
                                                 template<> struct ConwayPolynomial<73, 7> { using ZPZ = aerobus::zpz<73>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<68»; }; // NOLINT
 04572
                                                  template<> struct ConwayPolynomial<73, 8> { using ZPZ = aerobus::zpz<73>; using type =
                               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<53>, ZPZV<39>, ZPZV<18>, ZPZV<5»; }; //
                                                     template<> struct ConwayPolynomial<73, 9> { using ZPZ = aerobus::zpz<73>; using type 
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<15>, ZPZV<68»; };
                                 // NOLINT
04574
                                                  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>
                                                 template<> struct ConwayPolynomial<73, 11> { using ZPZ = aerobus::zpz<73>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<5>, ZPZV<68»; }; // NOLINT
   template<> struct ConwayPolynomial<73, 12> { using ZPZ = aerobus::zpz<73>; using type =
04576
                               POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<52>, ZPZV<26>, ZPZV<26>, ZPZV<46>, ZPZV<46>, ZPZV<29>, ZPZV<25>, ZPZV<25>, ZPZV<5>; }; // NOLINT
                                                     template<> struct ConwayPolynomial<73, 13> { using ZPZ = aerobus::zpz<73>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<68»; }; // NOLINT
template<> struct ConwayPolynomial<73, 15> { using ZPZ = aerobus::zpz<73>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<68»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<73, 19> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5</pre>, ZPZV<0>, ZP
 04581
                                                   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 =
04582
                               POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
 04583
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76»; }; // NOLINT template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<79, 5> { using ZPZ = aerobus::zpz<79>; using type =
 04585
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<76»; }; // NOLINT template<> struct ConwayPolynomial<79, 6> { using ZPZ = aerobus::zpz<79>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<68>, ZPZV<68>; };
                        template<> struct ConwayPolynomial<79, 7> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76»; }; // NOLI
                                        template<> struct ConwayPolynomial<79, 8> { using ZPZ = aerobus::zpz<79>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<5>, ZPZV<59>, ZPZV<48>, ZPZV<3»; }; //
                         template<> struct ConwayPolynomial<79, 9> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<57>, ZPZV<19>, ZPZV<76»; };
                         template<> struct ConwayPolynomial<79, 10> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<51>, ZPZV<1>, ZPZV<1>, ZPZV<30>, ZPZV<42>,
04590
                          ZPZV<3»; }; // NOLINT
                                           template<> struct ConwayPolynomial<79, 11> { using ZPZ = aerobus::zpz<79>; using type
04591
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<3>, ZPZV<76»; }; // NOLINT</pre>
04592
                                          template<> struct ConwayPolynomial<79, 12> { using ZPZ = aerobus::zpz<79>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<45>, ZPZV<52>, ZPZV<7>, ZPZV<40>,
                          ZPZV<59>, ZPZV<62>, ZPZV<3»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<78>, ZPZV<4>, ZPZV<76»; }; // NOLINT</pre>
04594
                                        template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04595
                                          template<> struct ConwayPolynomial<79, 19> { using ZPZ = aerobus::zpz<79>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<76»; };</pre>
                          NOLINT
04596
                                          template<> struct ConwayPolynomial<83, 1> { using ZPZ = aerobus::zpz<83>; using type =
                         POLYV<ZPZV<1>, ZPZV<81»; }; // NOLINT
                                         template<> struct ConwayPolynomial<83, 2> { using ZPZ = aerobus::zpz<83>; using type =
04597
                         POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT template<> struct ConwayPolynomial<83, 4> { using ZPZ = aerobus::zpz<83>; using type =
 04599
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<83, 5> { using ZPZ = aerobus::zpz<83>; using type =
04600
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<81»; }; // NOLINT
                                           template<> struct ConwayPolynomial<83, 6> { using ZPZ = aerobus::zpz<83>; using type =
 04601
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<76>, ZPZV<32>, ZPZV<17>, ZPZV<2»; }; // NOLINT
 04602
                                        template<> struct ConwayPolynomial<83, 7> { 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<3>, ZPZV<61»; }; // NOLINT template<> struct ConwayPolynomial<83, 8> { using ZPZ = aerobus::zpz<83>; using type =
 04603
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<23>, ZPZV<42>, ZPZV<42>, ZPZV<42>, ZPZV<40
                                         template<> struct ConwayPolynomial<83, 9> { using ZPZ = aerobus::zpz<83>; using type
 04604
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<24>, ZPZV<18>, ZPZV<81»; };
                          // NOLINT
04605
                                         template<> struct ConwayPolynomial<83, 10> { using ZPZ = aerobus::zpz<83>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<73>, ZPZV<53>,
                          ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<17>, ZPZV<81»; }; // NOLINT</pre>
04607
                                          template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35>, ZPZV<31>, ZPZV<31>, ZPZV<65>, ZPZV<55>, ZPZV<75>, ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         template<> struct ConwayPolynomial<83, 17> { using ZPZ = aerobus::zpz<83>; using type =
04609
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZ
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<81»; }; //</pre>
                          NOLINT
                                          template<> struct ConwayPolynomial<89, 1> { using ZPZ = aerobus::zpz<89>; using type =
04611
                         POLYV<ZPZV<1>, ZPZV<86»; }; // NOLINT
                                           template<> struct ConwayPolynomial<89, 2> { using ZPZ = aerobus::zpz<89>; using type =
                          POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; }; // NOLINT
 04613
                                        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 =
 04614
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<89, 5> { using ZPZ = aerobus::zpz<89>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<86»; }; // NOLINT
 04616
                                        template<> struct ConwayPolynomial<89, 6> { using ZPZ = aerobus::zpz<89>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<82>, ZPZV<80>, ZPZV<15>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type =
 04617
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<86»; }; // NOLINT
                                           template<> struct ConwayPolynomial<89, 8> { using ZPZ = aerobus::zpz<89>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<40>, ZPZV<79>, ZPZV<3»; };
                          NOLINT
04619
                                        template<> struct ConwayPolynomial<89, 9> { using ZPZ = aerobus::zpz<89>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<12>, ZPZV<12>, ZPZV<66, ZPZV<86»; };
                           // NOLINT
```

```
template<> struct ConwayPolynomial<89, 10> { using ZPZ = aerobus::zpz<89>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<33>, ZPZV<82>, ZPZV<82>, ZPZV<44>,
                           ZPZV<3»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                                                                                                                     // NOLINT
                           ZPZV<26>, ZPZV<86»; };
                          template<> struct ConwayPolynomial<89, 12> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<85>, ZPZV<15>, ZPZV<44>, ZPZV<51>, ZPZV<8>,
                           ZPZV<70>, ZPZV<52>, ZPZV<3»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                            template<> struct ConwayPolynomial<89, 17> { using ZPZ = aerobus::zpz<89>; using type
04624
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<26</pre>, }; // NOLINT
                          template<> struct ConwayPolynomial<89, 19> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04625
                                             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 =
04627
                          POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5»; }; // NOLINT
                                            template<> struct ConwayPolynomial<97, 3> { using ZPZ = aerobus::zpz<97>; using type =
04628
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<92»; }; // NOLINT
                                             template<> struct ConwayPolynomial<97, 4> { using ZPZ = aerobus::zpz<97>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; }; // NOLINT
04630
                                            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
                         template<> struct ConwayPolynomial<97, 6> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<58>, ZPZV<88>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<97, 7> { using ZPZ = aerobus::zpz<97>; using type =
04631
04632
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5
04633
                                         template<> struct ConwayPolynomial<97, 8> { using ZPZ = aerobus::zpz<97>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<55>, ZPZV<1>, ZPZV<32>, ZPZV<55>; }; template<> struct ConwayPolynomial<97, 9> { using ZPZ = aerobus::zpz<97>; using type =
04634
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<59>, ZPZV<59>, ZPZV<72, ZPZV<92»; };
                                            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<20>,
                           ZPZV<5»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
04636
                          template<> struct ConwayPolynomial<97, 12> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<59>, ZPZV<81>, ZPZV<86>, ZPZV<86>,
                           ZPZV<78>, ZPZV<94>, ZPZV<5»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<97, 13> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                             template<> struct ConwayPolynomial<97, 17> { using ZPZ = aerobus::zpz<97>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<9>; }; // NOLINT
                          template<> struct ConwayPolynomial<97, 19> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>
                                            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 =
04642
                         POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
04643
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<99»; }; // NOLINT
                                              template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<78>, ZPZV<2»; }; // NOLINT
04645
                                           template<> struct ConwayPolynomial<101, 5> { using ZPZ = aerobus::zpz<101>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<9»; }; // NOLINT template<> struct ConwayPolynomial<101, 6> { using ZPZ = aerobus::zpz<101>; using type =
04646
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<90>, ZPZV<20>, ZPZV<67>, ZPZV<2»; }; // NOLINT
04647
                                             template<> struct ConwayPolynomial<101, 7> { using ZPZ = aerobus::zpz<101>; using type
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<99»; }; // NOLINT
04648
                                         template<> struct ConwayPolynomial<101, 8> { using ZPZ = aerobus::zpz<101>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76>, ZPZV<29>, ZPZV<24>, ZPZV<24>, ZPZV<2»; }; //
                           NOLINT
                                           template<> struct ConwayPolynomial<101, 9> { using ZPZ = aerobus::zpz<101>; using type
04649
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64>, ZPZV<47>, ZPZV<99»; };
04650
                                           \texttt{template<> struct ConwayPolynomial<101, 10> \{ using ZPZ = \verb|aerobus::zpz<101>; using type| }
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<67>, ZPZV<49>, ZPZV<100>, ZPZV<100>, ZPZV<52>,
                           ZPZV<2»: 1: // NOLINT
                                             template<> struct ConwayPolynomial<101, 11> { using ZPZ = aerobus::zpz<101>; using type :
04651
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                           ZPZV<31>, ZPZV<99»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<101, 12> { using ZPZ = aerobus::zpz<101>; using type
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<79>, ZPZV<64>, ZPZV<39>, ZPZV<78>, ZPZV<48>, ZPZV<84>, ZPZV<21>, ZPZV<22>; }; // NOLINT
04653
                                         template<> struct ConwayPolynomial<101, 13> { 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>,
                                               V<0>, ZPZV<0>, ZPZV<7>, ZPZV<99»; }; // NOLINT
template<> struct ConwayPolynomial<101, 17> { using ZPZ = aerobus::zpz<101>; using type =
                               ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<99»; };</pre>
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<99»; }; // NOLINT
template<> struct ConwayPolynomial<101, 19> { using ZPZ = aerobus::zpz<101>; using type =
04655
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<29»; };</pre>
                                                  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 =
04657
                              POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<103, 3> { using ZPZ = aerobus::zpz<103>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<98»; }; // NOLINT
                                                 \texttt{template<>} \ \texttt{struct ConwayPolynomial<103, 4> \{ \ using \ \mathtt{ZPZ} = \mathtt{aerobus::zpz<103>; using \ type = 1.5 } \}
 04659
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<88>, ZPZV<5>; }; // NOLINT template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
04660
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<98»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<103, 6> { using ZPZ = aerobus::zpz<103>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<9>, ZPZV<30>, ZPZV<5»; }; // NOLINT
 04662
                                                   template<> struct ConwayPolynomial<103, 7> { using ZPZ = aerobus::zpz<103>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<98»; }; // NOLINT template<> struct ConwayPolynomial<103, 8> { using ZPZ = aerobus::zpz<103>; using type =
 04663
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<70>, ZPZV<71>, ZPZV<71>, ZPZV<4>; ; ;
                                                template<> struct ConwayPolynomial<103, 9> { using ZPZ = aerobus::zpz<103>; using type =
04664
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<51>, ZPZV<98»; };
                               // NOLINT
04665
                                                  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 =
04666
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<5>, ZPZV<98»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<103, 12> { using ZPZ = aerobus::zpz<103>; using type =
04667
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<74>, ZPZV<23>, ZPZV<94>, ZPZV<20>, ZPZV<81>, ZPZV<29>, ZPZV<88>, ZPZV<5»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<103, 13> { using ZPZ = aerobus::zpz<103>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98»; };</pre>
                                                                                                                                                                                                                                           // NOLINT
                              template<> struct ConwayPolynomial<103, 17> { using ZPZ = aerobus::zpz<103>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04669
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<102>, ZPZV<8>, ZPZV<98»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<103, 19> { using ZPZ = aerobus::zpz<103>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0</pre>, ZPZV<0>, ZP
                               NOLINT
04671
                                                  template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
                              POLYV<ZPZV<1>, ZPZV<105»; }; // NOLINT
04672
                                                   template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
                              POLYV<ZPZV<1>, ZPZV<103>, ZPZV<2»; }; // NOLINT
 04673
                                                   template<> struct ConwayPolynomial<107, 3> { using ZPZ = aerobus::zpz<107>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<105»; }; // NOLINT template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
 04674
                             POLYVCZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<107, 5> { using ZPZ = aerobus::zpz<107>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<105»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<107, 6> { using ZPZ = aerobus::zpz<107>; using type =
 04676
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<52>, ZPZV<22>, ZPZV<79>, ZPZV<2»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type =
04677
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<105»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<107, 8> { using ZPZ = aerobus::zpz<107>; using type =
                               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<24>, ZPZV<95>, ZPZV<2»; };
04679
                                               template<> struct ConwayPolynomial<107, 9> { using ZPZ = aerobus::zpz<107>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<66>, ZPZV<105»; };
                                // NOLINT
                                                  template<> struct ConwayPolynomial<107, 10> { using ZPZ = aerobus::zpz<107>; using type =
04680
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<94>, ZPZV<61>, ZPZV<83>, ZPZV<83>, ZPZV<95>,
                                ZPZV<2»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<107, 11> { using ZPZ = aerobus::zpz<107>; using type =
04681
                               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
04682
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<61>, ZPZV<61>, ZPZV<42>, ZPZV<57>, ZPZV<57>, ZPZV<57>, ZPZV<58, ZPZV<58, ZPZV<58, ZPZV<69, ZPZV<61>, ZPZV<61>, ZPZV<62, ZPZV<65, ZPZV<65, ZPZV<65, ZPZV<61>, ZPZV<65, ZPZV<6
04683
                                                 template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                   template<> struct ConwayPolynomial<107, 17> { using ZPZ = aerobus::zpz<107>; using type =
04684
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<105»; }; // NOLINT
template<> struct ConwayPolynomial<107, 19> { using ZPZ = aerobus::zpz<107>; using type =
04685
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<105»; }; //
                               NOLTNT
```

```
04686
                                             template<> struct ConwayPolynomial<109, 1> { using ZPZ = aerobus::zpz<109>; using type =
                          POLYV<ZPZV<1>, ZPZV<103»; }; // NOLINT
04687
                                          template<> struct ConwayPolynomial<109, 2> { using ZPZ = aerobus::zpz<109>; using type =
                          POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; }; // NOLINT
 04688
                                            template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
                         POLYVCZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; }; // NOLINT
 04690
                                          template<> struct ConwayPolynomial<109, 5> { using ZPZ = aerobus::zpz<109>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103»; }; // NOLINT
                                            template<> struct ConwayPolynomial<109, 6> { using ZPZ = aerobus::zpz<109>; using type =
04691
                          04692
                                            template<> struct ConwayPolynomial<109,
                                                                                                                                                                                                                       7> { using ZPZ = aerobus::zpz<109>; using type
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<143, ZPZV<103»; }; //
 04693
                                         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»; };
                           NOLTNT
                          template<> struct ConwayPolynomial<109, 9> { using ZPZ = aerobus::zpz<109>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 
04694
                                           template<> struct ConwayPolynomial<109, 10> { using ZPZ = aerobus::zpz<109>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<75>, ZPZV<55>, ZPZV<16>, ZPZV<16>, ZPZV<69>,
                           ZPZV<6»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<109, 11> { using ZPZ = aerobus::zpz<109>; using type =
04696
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<11>, ZPZV<103»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<109, 12> { using ZPZ = aerobus::zpz<109>; using type =
04697
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<50>, ZPZV<55>, ZPZV<53>, ZPZV<37>, ZPZV<85, ZPZV<65>,
                           ZPZV<103>, ZPZV<28>, ZPZV<6»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<109, 13> { using ZPZ = aerobus::zpz<109>; using type =
04698
                           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 =
04699
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           04700
                           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<15>, ZPZV<15</pre>
                                             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 =
04702
                          POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3»: }: // NOLINT
                                           template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<110»; }; // NOLINT template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
 04704
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<62>, ZPZV<3»; }; // NOLINT
                                          \texttt{template<>} \texttt{struct ConwayPolynomial<113, 5> \{ \texttt{using ZPZ = aerobus::zpz<113>; using type = 1.5 \}}
 04705
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110»; }; // NOLINT template<> struct ConwayPolynomial<113, 6> { using ZPZ = aerobus::zpz<113>; using type =
 04706
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<59>, ZPZV<30>, ZPZV<71>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<113, 7> { using ZPZ = aerobus::zpz<113>; using type =
 04707
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
04708
                                          template<> struct ConwayPolynomial<113, 8> { using ZPZ = aerobus::zpz<113>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<98>, ZPZV<38>, ZPZV<28>, ZPZV<28>, ZPZV<3»; }; //
                          NOLINT
                                            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<87>, ZPZV<87>, ZPZV<11>, ZPZV<110»; };
04710
                                           template<> struct ConwayPolynomial<113, 10> { using ZPZ = aerobus::zpz<113>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<108>, ZPZV<57>, ZPZV<45>, ZPZV<83>, ZPZV<56>,
                           ZPZV<3»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04712
                                         template<> struct ConwayPolynomial<113, 12> { using ZPZ = aerobus::zpz<113>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<23>, ZPZV<62>, ZPZV<4>, ZPZV<48>, ZPZV<56>,
                           ZPZV<10>, ZPZV<27>, ZPZV<3»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<113, 13> { using ZPZ = aerobus::zpz<113>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110»; }; // NOLINT</pre>
04714
                                         template<> struct ConwayPolynomial<113, 17> { using ZPZ = aerobus::zpz<113>; using type =
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
04715
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2</pre>
                                           POLYV<ZPZV<1>, ZPZV<124»; }; // NOLINT
                                            template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
 04717
                         POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124»; };
                                                                                                                                                                                                                                          // NOLINT
                        template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<97>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<127, 5> { using ZPZ = aerobus::zpz<127>; using type =
 04719
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<124»; };
                                         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 =
04722
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<124»; }; // NOLINT template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type =
04723
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<104>, ZPZV<55>, ZPZV<8>, ZPZV<8>, ZPZV<3»; }; //
                                       template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type
04724
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12+»;
                        }; // NOLINT
04725
                                         template<> struct ConwayPolynomial<127, 10> { using ZPZ = aerobus::zpz<127>; using type :
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<64>, ZPZV<95>, ZPZV<66>, ZPZV<4>,
                         ZPZV<3»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        \text{ZPZV}<11>, \text{ZPZV}<124»; }; // NOLINT
04727
                                       template<> struct ConwayPolynomial<127, 12> { using ZPZ = aerobus::zpz<127>; using type :
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<33>, ZPZV<97>, ZPZV<15>,
                                                                                                                                                        // NOLINT
                        ZPZV<99>, ZPZV<8>, ZPZV<3»; };</pre>
                                       template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<124»; }; // NOLINT
template<> struct ConwayPolynomial<127, 17> { using ZPZ = aerobus::zpz<127>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<124»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<127,
                                                                                                                                                                                                        19> { using ZPZ = aerobus::zpz<127>; using type =
04730
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<124»; }; //</pre>
                        NOLINT
                                       template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
04731
                       POLYV<ZPZV<1>, ZPZV<129»; }; // NOLINT
                                        template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
                       POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2»; }; // NOLINT
04733
                                        template<> struct ConwayPolynomial<131, 3> { using ZPZ = aerobus::zpz<131>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<129»; }; // NOLINT
template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2»; }; // NOLINT
04734
                                         template<> struct ConwayPolynomial<131, 5> { using ZPZ = aerobus::zpz<131>; using type =
04735
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<129»; }; // NOLINT
04736
                                      template<> struct ConwayPolynomial<131, 6> { using ZPZ = aerobus::zpz<131>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<4>, ZPZV<22>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<131, 7> { using ZPZ = aerobus::zpz<131>; using type =
04737
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12, ZPZV<129%; }; // 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»; };
                                     template<> struct ConwayPolynomial<131, 9> { using ZPZ = aerobus::zpz<131>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<19>, ZPZV<129»; };
                        // NOLINT
                                        template<> struct ConwayPolynomial<131, 10> { using ZPZ = aerobus::zpz<131>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<124>, ZPZV<97>, ZPZV<9>, ZPZV<126>, ZPZV<44>,
                        ZPZV<2»; }; // NOLINT</pre>
04741
                                        template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<6>, ZPZV<129»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<131, 12> { using ZPZ = aerobus::zpz<131>; using type
                        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>
04743
                                         template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type :
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        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<6>, ZPZV<6>, ZPZV<6>, ZPZV<129»; };</pre>
                       template<> struct ConwayPolynomial<131, 19> { using ZPZ = aerobus::zpz<131>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04745
                       NOLINT
04746
                                         template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
                       POLYV<ZPZV<1>, ZPZV<134»; }; // NOLINT
04747
                                      template<> struct ConwayPolynomial<137, 2> { using ZPZ = aerobus::zpz<137>; using type =
                       POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type =
04748
                       POLYV<ZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<95>, ZPZV<3»; }; // NOLINT
04750
                                        template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<134»; }; // NOLINT
                                       template<> struct ConwayPolynomial<137, 6> { using ZPZ = aerobus::zpz<137>; using type =
04751
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<116>, ZPZV<102>, ZPZV<3>, ZPZV<3»; }; // NOLINT
                                        template<> struct ConwayPolynomial<137,
                                                                                                                                                                                                        7> { using ZPZ = aerobus::zpz<137>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<134w; }; // NoLII template<> struct ConwayPolynomial<137, 8> { using ZPZ = aerobus::zpz<137>; using type = aerobus::zpz<137>; usin
04753
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105>, ZPZV<21>, ZPZV<34>, ZPZV<3*; };
04754
                                       template<> struct ConwayPolynomial<137, 9> { using ZPZ = aerobus::zpz<137>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<80>, ZPZV<80>, ZPZV<134»;
                      }; // NOLINT
                                    template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<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»; };</pre>
                                                                                                               // NOLINT
                                   template<> struct ConwayPolynomial<137, 12> { using ZPZ = aerobus::zpz<137>; using type =
04757
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<61>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<36>,
                      ZPZV<135>, ZPZV<61>, ZPZV<3»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<137, 13> { using ZPZ = aerobus::zpz<137>; using type
04758
                      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
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04760
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<134»; }; //</pre>
04761
                                     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 =
04762
                      POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2»; }; // NOLINT
                                     template<> struct ConwayPolynomial<139, 3> { using ZPZ = aerobus::zpz<139>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<137»; }; // NOLINT
                                    template<> struct ConwayPolynomial<139, 4> { using ZPZ = aerobus::zpz<139>; using type =
04764
                      template<> struct ConwayPolynomial<139, 5> { using ZPZ = aerobus::zpz<139>; using type =
04765
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<137»; }; // NOLINT
04766
                                      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
04767
                                   template<> struct ConwayPolynomial<139, 7> { using ZPZ = aerobus::zpz<139>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<137»; }; // NOLINT template<> struct ConwayPolynomial<139, 8> { using ZPZ = aerobus::zpz<139>; using type =
04768
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>, ZPZV<36>, ZPZV<21>, ZPZV<2»; }; //
                                     template<> struct ConwayPolynomial<139, 9> { using ZPZ = aerobus::zpz<139>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<70>, ZPZV<87>, ZPZV<137»; };
                       // NOLINT
                     template<> struct ConwayPolynomial<139, 10> { using ZPZ = aerobus::zpz<139>; using type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<110>, ZPZV<48>, ZPZV<130>, ZPZV<66>, ZPZV<110>, ZPZV<110, ZPZV<110
04770
                      ZPZV<106>, ZPZV<2»; };</pre>
                                                                                                               // NOLINT
                                     template<> struct ConwayPolynomial<139, 11> { using ZPZ = aerobus::zpz<139>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<7>, ZPZV<137»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<139, 12> { using ZPZ = aerobus::zpz<139>; using type :
                      POLYV<ZPZV<10>, ZPZV<0>, ZPZV<2»; }; // NOLINT
                                      template<> struct ConwayPolynomial<139, 13> { using ZPZ = aerobus::zpz<139>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<137»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type =
04774
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137»; }; // NOLINT
template<> struct ConwayPolynomial<139, 19> { using ZPZ = aerobus::zpz<139>; using type
                      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<23>, ZPZV<23>, ZPZV<137»; }; //</pre>
                      NOLINT
04776
                                    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
04778
                                   template<> struct ConwayPolynomial<149, 3> { using ZPZ = aerobus::zpz<149>; using type =
                    POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147»; }; // NOLINT
template<> struct ConwayPolynomial<1149, 4> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<20*; }; // NOLINT
template<> struct ConwayPolynomial<149, 5> { using ZPZ = aerobus::zpz<149>; using type =
04779
04780
                    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 =
04781
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<105>, ZPZV<33>, ZPZV<55>, ZPZV<2»; }; // NOLINT
                                     template<> struct ConwayPolynomial<149, 7> { using ZPZ = aerobus::zpz<149>; using type
04782
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 8> { using ZPZ = aerobus::zpz<149>; using type =
04783
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<140>, ZPZV<25>, ZPZV<123>, ZPZV<123>, ZPZV<2); };
                      NOLINT
04784
                                   template<> struct ConwayPolynomial<149, 9> { using ZPZ = aerobus::zpz<149>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<146>, ZPZV<20>, ZPZV<147»;
                      }; // NOLINT
04785
                                      template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<74>, ZPZV<42>, ZPZV<148>, ZPZV<143>, ZPZV<51>,
                      ZPZV<2»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<149, 11> { using ZPZ = aerobus::zpz<149>; using type :
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147»; }; // NOLINT
                                   template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<91>, ZPZV<52>, ZPZV<9>,
                       ZPZV<104>, ZPZV<110>, ZPZV<2»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<149, 13> { using ZPZ = aerobus::zpz<149>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<147»; };</pre>
                                                                                                                                                                                       // NOLINT
                                     template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type =
04789
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       template<> struct ConwayPolynomial<149, 19> { using ZPZ = aerobus::zpz<149>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       NOLINT
04791
                                      template<> struct ConwayPolynomial<151, 1> { using ZPZ = aerobus::zpz<151>; using type =
                       POLYV<ZPZV<1>, ZPZV<145»; }; // NOLINT
04792
                                    template<> struct ConwayPolynomial<151, 2> { using ZPZ = aerobus::zpz<151>; using type =
                      POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<151, 3> { using ZPZ = aerobus::zpz<151>; using type =
04793
                      POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<145»; }; // NOLINT template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6»; }; // NOLINT
                                       template<> struct ConwayPolynomial<151, 5> { using ZPZ = aerobus::zpz<151>; using type =
04795
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<145»; }; // NOLINT
04796
                                     template<> struct ConwayPolynomial<151, 6> { using ZPZ = aerobus::zpz<151>; using type =
                     template<> struct ConwayFolynomial<151, 0> { using ZF2 - aerobus.:ZPZV151>, using type POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<125>, ZPZV<15>, ZPZV<6»; }; // NOLINT template<> struct ConwayFolynomial<151, 7> { using ZPZ = aerobus::zpz<151>; using type
04797
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<145»; }; //
                                     template<> struct ConwayPolynomial<151, 8> { using ZPZ = aerobus::zpz<151>; using type
04798
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<140>, ZPZV<122>, ZPZV<43>, ZPZV<6»; };
                       NOLINT
04799
                                    template<> struct ConwayPolynomial<151, 9> { using ZPZ = aerobus::zpz<151>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<126>, ZPZV<16>, ZPZV<96>, ZPZV<145»;
                      }; // NOLINT template<> struct ConwayPolynomial<151, 10> { using ZPZ = aerobus::zpz<151>; using type
04800
                       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>
04801
                                      template<> struct ConwayPolynomial<151, 11> { using ZPZ = aerobus::zpz<151>; using type :
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<145»; }; // NOLINT
                                       template<> struct ConwayPolynomial<151, 12> { using ZPZ = aerobus::zpz<151>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<121>, ZPZV<121>, ZPZV<101>, ZPZV<6>, ZPZV<77>,
                       ZPZV<107>, ZPZV<147>, ZPZV<6»; }; // NOLINT</pre>
04803
                                     template<> struct ConwayPolynomial<151, 13> { using ZPZ = aerobus::zpz<151>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<145»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<151, 17> { using ZPZ = aerobus::zpz<151>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       template<> struct ConwayPolynomial<151, 19> { using ZPZ = aerobus::zpz<151>; using type =
                      POLYV<ZPZV<0>, ZPZV<0>, ZPZV<1, ZPZV<0>, ZPZV<0>
                       NOLINT
                                       template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
                      POLYV<ZPZV<1>, ZPZV<152»; }; // NOLINT template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
04807
                      POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<157, 3> { using ZPZ = aerobus::zpz<157>; using type =
04808
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<152»; }; // NOLINT
                                   template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
04809
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<136, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<157, 5> { using ZPZ = aerobus::zpz<157>; using type =
04810
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>; }; // NOLINT template<> struct ConwayPolynomial<157, 6> { using ZPZ = aerobus::zpz<157>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<130>, ZPZV<43>, ZPZV<144>, ZPZV<5»; }; // NOLINT
04811
                                       template<> struct ConwayPolynomial<157, 7> { using ZPZ = aerobus::zpz<157>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<14>, ZPZV<152»; };
                      template<> struct ConwayPolynomial<157, 8> { using ZPZ = aerobus::zpz<157>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<40>, ZPZV<5»; }; //
04813
                      NOLINT
                                     template<> struct ConwayPolynomial<157, 9> { using ZPZ = aerobus::zpz<157>; using type =
04814
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<114>, ZPZV<52>, ZPZV<152»;
                       }; // NOLINT
04815
                                     template<> struct ConwayPolynomial<157, 10> { using ZPZ = aerobus::zpz<157>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<61>, ZPZV<22>, ZPZV<124>, ZPZV<124>, ZPZV<61>, ZPZV<93>,
                       ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<157, 11> { using ZPZ = aerobus::zpz<157>; using type =
04816
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<29>, ZPZV<152»; };</pre>
                                                                                                                  // NOLINT
04817
                                     template<> struct ConwayPolynomial<157, 12> { using ZPZ = aerobus::zpz<157>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<77>, ZPZV<110>, ZPZV<72>, ZPZV<137>, ZPZV<43>,
                      ZPZV<152>, ZPZV<57>, ZPZV<58; }; // NOLINT

template<> struct ConwayPolynomial<157, 13> { using ZPZ = aerobus::zpz<157>; using type :
04818
                        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<156>, ZPZV<9>, ZPZV<152»; }; // NOLINT
template<> struct ConwayPolynomial<157, 17> { using ZPZ = aerobus::zpz<157>; using type
04819
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
04820
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<14>, ZPZV<152»; }; //</pre>
                           NOLINT
04821
                                           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 =
04822
                          POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; }; // NOLINT
                                             template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
 04823
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<161»; }; // NOLINT
template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<163, 5> { using ZPZ = aerobus::zpz<163>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2»; }; // NOLINT
Template<> STRUCO STRUCCO STRUCCO
 04824
04825
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<161»; }; // NOLINT
                                             template<> struct ConwayPolynomial<163, 6> { using ZPZ = aerobus::zpz<163>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<25>, ZPZV<156>, ZPZV<2»; }; // NOLINT
 04827
                                            template<> struct ConwayPolynomial<163, 7> { using ZPZ = aerobus::zpz<163>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<161»; }; // NOLINT
                                          template<> struct ConwayPolynomial<163, 8> { using ZPZ = aerobus::zpz<163>; using type =
04828
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<132>, ZPZV<83>, ZPZV<6>, ZPZV<2»; }; //
                          template<> struct ConwayPolynomial<163, 9> { using ZPZ = aerobus::zpz<163>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<162>, ZPZV<167>, ZPZV<161>;
                           }; // NOLINT
                          template<> struct ConwayPolynomial<163, 10> { using ZPZ = aerobus::zpz<163>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<111>, ZPZV<120>, ZPZV<125>, ZPZV<15>, ZPZV<0>,
04830
                           ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<163, 11> { using ZPZ = aerobus::zpz<163>; using type =
04831
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<11>, ZPZV<161»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type =
04832
                          POLYY<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<2w; }; // NOLINT
                                             template<> struct ConwayPolynomial<163, 13> { using ZPZ = aerobus::zpz<163>; using type :
04833
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<24>, ZPZV<161»; }; // NOLINT
  template<> struct ConwayPolynomial<163, 17> { using ZPZ = aerobus::zpz<163>; using type =
04834
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
 04835
                                             template<> struct ConwayPolynomial<163, 19> { 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<0>, ZPZV<0>, ZPZV<8>, ZPZV<8</pre>, //
                           NOLINT
                                            template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
04836
                          POLYV<ZPZV<1>, ZPZV<162»; }; // NOLINT
                                             template<> struct ConwayPolynomial<167, 2> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<166>, ZPZV<5»; }; // NOLINT
                                          template<> struct ConwayPolynomial<167, 3> { using ZPZ = aerobus::zpz<167>; using type =
 04838
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162»; }; // NOLINT template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type =
04839
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 5> { using ZPZ = aerobus::zpz<167>; using type =
04840
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<162»; }; // NOLINT
 04841
                                             template<> struct ConwayPolynomial<167, 6> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<75>, ZPZV<38>, ZPZV<38; }; // NOLINT template<> struct ConwayPolynomial<167, 7> { using ZPZ = aerobus::zpz<167>; using type =
04842
                          POLYV<ZPZV<1>, ZPZV<0>, 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 =
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<149>, ZPZV<56>, ZPZV<113>, ZPZV<5»; };
04844
                                             template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<165>, ZPZV<162»;
                           }; // NOLINT
                                             template<> struct ConwayPolynomial<167, 10> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<85>, ZPZV<68>, ZPZV<109>, ZPZV<143>, ZPZV<148>, ZPZV<5»; }; // NOLINT
04846
                                         template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<24>, ZPZV<162»; }; // NOLINT
                                            template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type =
04847
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<142>, ZPZV<10>, ZPZV<141>, ZPZV<131>,
                           ZPZV<140>, ZPZV<41>, ZPZV<57>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 13> { using ZPZ = aerobus::zpz<167>; using type =
04848
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<162»; }; // NOLINT template<> struct ConwayPolynomial<167, 17> { using ZPZ = aerobus::zpz<167>; using type =
04849
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 32>, \ \texttt{ZPZV} < 162*; \ \}; \ \ // \ \texttt{NOLINT} 
04850
                                           template<> struct ConwayPolynomial<167, 19> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           NOLINT
                                             template<> struct ConwayPolynomial<173, 1> { using ZPZ = aerobus::zpz<173>; using type =
                          POLYV<ZPZV<1>, ZPZV<171»; }; // NOLINT
 04852
                                          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 =
 04853
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171»; }; // NOLINT
```

```
04854
                                             template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<20; }; // NOLINT template<> struct ConwayPolynomial<173, 5> { using ZPZ = aerobus::zpz<173>; using type =
04855
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171»; }; // NOLINT
                         template<> struct ConwayPolynomial<173, 6> { using ZPZ = aerobus::zpz<173>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<134>, ZPZV<107>, ZPZV<2»; }; // NOLINT</pre>
 04856
                                           template<> struct ConwayPolynomial<173, 7> { using ZPZ = aerobus::zpz<173>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<171»; };
                                         template<> struct ConwayPolynomial<173, 8> { using ZPZ = aerobus::zpz<173>; using type =
 04858
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<125>, ZPZV<158>, ZPZV<27>, ZPZV<27>, ZPZV<29; }; //
                         NOLINT
                                         template<> struct ConwayPolynomial<173, 9> { using ZPZ = aerobus::zpz<173>; using type =
04859
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<104>, ZPZV<171»;
                          }; // NOLINT
 04860
                                         template<> struct ConwayPolynomial<173, 10> { using ZPZ = aerobus::zpz<173>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<164>, ZPZV<164>, ZPZV<48>, ZPZV<106>, ZPZV<58>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<173, 11> { using ZPZ = aerobus::zpz<173>; using type =
04861
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                          ZPZV<12>, ZPZV<171»; };</pre>
                                                                                                                               // NOLINT
                                           template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<26>, ZPZV<46>, ZPZV<46>, ZPZV<46>, ZPZV<166>, ZPZV<0>,
                          ZPZV<159>, ZPZV<22>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<173, 13> { using ZPZ = aerobus::zpz<173>; using type =
04863
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171»; };</pre>
                                                                                                                                                                                                          // NOLINT
                                         template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type =
04864
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                          template<> struct ConwayPolynomial<173, 19> { using ZPZ = aerobus::zpz<173>; using type =
04865
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>
                                         template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<177»; }; // NOLINT
                                            template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
04867
                         POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<179, 3> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<177»; }; // NOLINT template<> struct ConwayPolynomial<179, 4> { using ZPZ = aerobus::zpz<179>; using type =
 04869
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<179, 5> { using ZPZ = aerobus::zpz<179>; using type =
04870
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<177»; }; // NOLINT
 04871
                                          template<> struct ConwayPolynomial<179, 6> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<91>, ZPZV<55>, ZPZV<109>, ZPZV<2»; }; // NOLINT
 04872
                                           template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<177»; }; // NOLINT template<> struct ConwayPolynomial<179, 8> { using ZPZ = aerobus::zpz<179>; using type =
 04873
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<163>, ZPZV<144>, ZPZV<73>, ZPZV<73>; };
                          NOLINT
04874
                                           template<> struct ConwayPolynomial<179, 9> { using ZPZ = aerobus::zpz<179>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<40
                           // NOLINT
04875
                                           template<> struct ConwayPolynomial<179, 10> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<150>, ZPZV<49>, ZPZV<87>,
                          ZPZV<2»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<179, 11> { using ZPZ = aerobus::zpz<179>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<28>, ZPZV<177»; }; // NOLINT</pre>
04877
                                            template<> struct ConwayPolynomial<179, 12> { using ZPZ = aerobus::zpz<179>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<103>, ZPZV<83>, ZPZV<43>, ZPZV<76>, ZPZV<8>, ZPZV<177>, ZPZV<1>, ZPZV<1>, ZPZV<2>; }; // NOLINT
04878
                                           template<> struct ConwayPolynomial<179,
                                                                                                                                                                                                                    13> { using ZPZ = aerobus::zpz<179>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                           // NOLINT
                          ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<177»; };</pre>
                                        template<> struct ConwayPolynomial<179, 17> { using ZPZ = aerobus::zpz<179>; using type =
04879
                          \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZ
04880
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177»; }; //</pre>
                         template<> struct ConwayPolynomial<181, 1> { using ZPZ = aerobus::zpz<181>; using type = POLYV<ZPZV<1>, ZPZV<179w; }; // NOLINT
04881
                                           template<> struct ConwayPolynomial<181, 2> { using ZPZ = aerobus::zpz<181>; using type =
04882
                         POLYV<ZPZV<1>, ZPZV<177>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
 04883
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179»; }; // NOLINT template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
04884
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<105>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
 04885
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<179»; }; // NOLINT
                                            template<> struct ConwayPolynomial<181, 6> { using ZPZ = aerobus::zpz<181>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<163>, ZPZV<169>, ZPZV<2»; }; // NOLINT
 04887
                                        template<> struct ConwayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1P; // NOLINT template<> struct ConwayPolynomial<181, 8> { using ZPZ = aerobus::zpz<181>; using type =
 04888
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<108>, ZPZV<22>, ZPZV<149>, ZPZV<2*; }; //
04889
                                   template<> struct ConwayPolynomial<181, 9> { using ZPZ = aerobus::zpz<181>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<107>, ZPZV<168>, ZPZV<179»;
                      }; // NOLINT
                                      template<> struct ConwayPolynomial<181, 10> { using ZPZ = aerobus::zpz<181>; using type =
04890
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<154>, ZPZV<104>, ZPZV<94>, ZPZV<95>, ZPZV<88>,
                      ZPZV<2»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<181, 11> { using ZPZ = aerobus::zpz<181>; using type =
04891
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04892
                                    template<> struct ConwayPolynomial<181, 12> { using ZPZ = aerobus::zpz<181>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<141>, ZPZV<45>, ZPZV<122>, ZPZV<175>, ZPZV<12>, ZPZV<10>, ZPZV<2»; }; // NOLINT
04893
                                  template<> struct ConwayPolynomial<181, 13> { using ZPZ = aerobus::zpz<181>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    template<> struct ConwayPolynomial<181, 17> { using ZPZ = aerobus::zpz<181>; using type =
04894
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<179»; };</pre>
                                    template<> struct ConwayPolynomial<181, 19> { using ZPZ = aerobus::zpz<181>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<179»; }; //</pre>
                      NOLINT
04896
                                     template<> struct ConwayPolynomial<191, 1> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<172»; }; // NOLINT
04897
                                    template<> struct ConwayPolynomial<191, 2> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19»; }; // NOLINT
04898
                                     template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172»; }; // NOLINT
template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
04899
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; };
                                                                                                                                                                                                                                              // NOLINT
                                     template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
04900
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<172»; }; // NOLINT
04901
                                     template<> struct ConwayPolynomial<191, 6> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<110>, ZPZV<10>, ZPZV<10>, ZPZV<19»; }; // NOLINT
                                    template<> struct ConwayPolynomial<191, 7> { using ZPZ = aerobus::zpz<191>; using type
04902
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<14>, ZPZV<172»; }; // NOLINT
                                     template<> struct ConwayPolynomial<191, 8> { using ZPZ = aerobus::zpz<191>, using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<164>, ZPZV<139>, ZPZV<171>, ZPZV<19»; }; //
                      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<0>, ZPZV<12+, ZPZV<124>, ZPZV<124>;
04904
                      }; // NoLINT
    template<> struct ConwayPolynomial<191, 10> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<113>, ZPZV<47>, ZPZV<173>, ZPZV<74>,
                      ZPZV<156>, ZPZV<19»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<191, 11> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<191, 12> { using ZPZ = aerobus::zpz<191>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<76>, ZPZV<168>, ZPZV<25>, ZPZV<49>, ZPZV<90>,
                       ZPZV<7>, ZPZV<151>, ZPZV<19»; }; // NOLINT</pre>
04908
                                    template<> struct ConwayPolynomial<191, 13> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<172»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<191, 17> { using ZPZ = aerobus::zpz<191>; using type
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<172»; };</pre>
                                                                                                                                                                                                                                                                                                                       // NOLINT
04910
                                     template<> struct ConwayPolynomial<191, 19> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                      NOLINT
                                      template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<188»; }; // NOLINT
04912
                                    template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5»; }; // NOLINT
                                    template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
04913
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; }; // NOLINT
                                      template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
04914
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<5»; }; // NOLINT
                                   template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
04915
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 6> { using ZPZ = aerobus::zpz<193>; using type =
04916
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<5>, ZPZV<172>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<193, 7> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<188»; };
04918
                                   template<> struct ConwayPolynomial<193, 8> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<145>, ZPZV<34>, ZPZV<154>, ZPZV<154>, ZPZV<5»; }; //
                      NOLINT
                                     template<> struct ConwayPolynomial<193, 9> { using ZPZ = aerobus::zpz<193>; using type
04919
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<168>, ZPZV<168>, ZPZV<27>, ZPZV<188»;
                      }; // NOLINT
04920
                                    template<> struct ConwayPolynomial<193, 10> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<51>, ZPZV<77>, ZPZV<89>, ZPZV<5»; }; // NOLINT
04921
                                  template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                             ZPZV<1>, ZPZV<188»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<193, 12> { using ZPZ = aerobus::zpz<193>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<155>, ZPZV<52>, ZPZV<135>, ZPŽV<152>,
                             ZPZV<90>, ZPZV<46>, ZPZV<28>, ZPZV<5»; };</pre>
                                                /<90>, ZPZV<46>, ZPZV<28>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<193, 13> { using ZPZ = aerobus::zpz<193>; using type =
04923
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<188»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<193,
                                                                                                                                                                                                                                                 17> { using ZPZ = aerobus::zpz<193>; using type =
04924
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                                                                                                 ZPZV<0>,
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<188»; }; // NOLINT
template<> struct ConwayPolynomial<193, 19> { using ZPZ = aerobus::zpz<193>; using type =
04925
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>
                             NOLINT
04926
                                                template<> struct ConwayPolynomial<197, 1> { using ZPZ = aerobus::zpz<197>; using type =
                            POLYV<ZPZV<1>, ZPZV<195»; }; // NOLINT
                                                template<> struct ConwayPolynomial<197, 2> { using ZPZ = aerobus::zpz<197>; using type =
04927
                             POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2»; }; // NOLINT
                                                template<> struct ConwayPolynomial<197, 3> { using ZPZ = aerobus::zpz<197>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<195»; }; // NOLINT template<> struct ConwayPolynomial<197, 4> { using ZPZ = aerobus::zpz<197>; using type =
04929
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<124>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<197, 5> { using ZPZ = aerobus::zpz<197>; using type =
04930
                            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 =
04932
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 >, ZPZV
                                               template<> struct ConwayPolynomial<197, 8> { using ZPZ = aerobus::zpz<197>; using type =
04933
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<176>, ZPZV<96>, ZPZV<29>, ZPZV<2»; }; //
                            NOLINT
                                                template<> struct ConwayPolynomial<197, 9> { using ZPZ = aerobus::zpz<197>; using type =
04934
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<127>, ZPZV<8>, ZPZV<195»;
                             }; // NOLINT
                                                template<> struct ConwayPolynomial<197, 10> { using ZPZ = aerobus::zpz<197>; using type :
04935
                             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>
                                                template<> struct ConwayPolynomial<197, 11> { using ZPZ = aerobus::zpz<197>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<14>, ZPZV<195»; }; // NOLINT</pre>
                            \label{eq:convergence} template<> struct ConwayPolynomial<197, 12> \{ using ZPZ = aerobus::zpz<197>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<168>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>, ZPZV<10+, ZPZV<141>, ZPZ
04937
                             ZPZV<90>, ZPZV<163>, ZPZV<2»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<197, 13> { using ZPZ = aerobus::zpz<197>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<195»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<197, 17> { using ZPZ = aerobus::zpz<197>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<195»; }; // NOLINT</pre>
                                                                                                                                                                                                                                                 19> { using ZPZ = aerobus::zpz<197>; using type
                                                template<> struct ConwayPolynomial<197,
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0</pre>; };
                             NOLINT
04941
                                                template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
                            POLYV<ZPZV<1>, ZPZV<196»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<199, 2> { using ZPZ = aerobus::zpz<199>; using type =
                            POLYV<ZPZV<1>, ZPZV<193>, ZPZV<3»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<199, 3> { using ZPZ = aerobus::zpz<199>; using type =
04943
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; }; // NOLINT template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
04944
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3; }; // NOLINT
template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
04945
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
04946
                                               template<> struct ConwayPolynomial<199, 6> { using ZPZ = aerobus::zpz<199>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<90>, ZPZV<58>, ZPZV<79>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<199, 7> { using ZPZ = aerobus::zpz<199>; using type
04947
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
                                               template<> struct ConwayPolynomial<199, 8> { using ZPZ = aerobus::zpz<199>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<160>, ZPZV<23>, ZPZV<159>, ZPZV<3»; };
04949
                                             template<> struct ConwayPolynomial<199, 9> { using ZPZ = aerobus::zpz<199>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<177>, ZPZV<141>, ZPZV<196»;
                             }; // NOLINT
                                                 template<> struct ConwayPolynomial<199, 10> { using ZPZ = aerobus::zpz<199>; using type =
04950
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<158>, ZPZV<31>, ZPZV<54>, ZPZV<9>,
                             ZPZV<3»; }; // NOLINT</pre>
04951
                                                template<> struct ConwayPolynomial<199, 11> { using ZPZ = aerobus::zpz<199>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<1>, ZPZV<196»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<199, 12> { using ZPZ = aerobus::zpz<199>; using type =
04952
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, 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 :
04953
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<196»; }; // NOLINT
template<> struct ConwayPolynomial<199, 17> { using ZPZ = aerobus::zpz<199>; using type =
04954
```

```
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<13>, ZPZV<196»; }; // NOLINT
template<> struct ConwayPolynomial<199, 19> { using ZPZ = aerobus::zpz<199>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19*, ZPZV<196*; }; //</pre>
                          NOLINT
                                          template<> struct ConwayPolynomial<211, 1> { using ZPZ = aerobus::zpz<211>; using type =
                         POLYV<ZPZV<1>, ZPZV<209»; }; // NOLINT
                                        template<> struct ConwayPolynomial<211, 2> { using ZPZ = aerobus::zpz<211>; using type =
                       POLYV<ZPZV<1>, ZPZV<207>, ZPZV<20x; }; // NOLINT template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
04958
                         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<2>, ZPZV<209»; }; // NOLINT template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
04959
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2»; }; // NOLINT
 04960
                                        template<> struct ConwayPolynomial<211, 5> { using ZPZ = aerobus::zpz<211>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<20, ZPZV<2>, ZPZV<20, ZPZV<20, ZPZV<20, ZPZV<20, ZPZV<20, ZPZV<20, ZPZV<20), ZPZV<20, ZPZV<
04961
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<209»; };
                                         template<> struct ConwayPolynomial<211, 8> { using ZPZ = aerobus::zpz<211>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<200>, ZPZV<87>, ZPZV<29>, ZPZV<2»; };
                                        template<> struct ConwayPolynomial<211, 9> { using ZPZ = aerobus::zpz<211>; using type =
04964
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<139>, ZPZV<139>, ZPZV<26>, ZPZV<209»;
                          }; // NOLINT
                                        template<> struct ConwayPolynomial<211, 10> { using ZPZ = aerobus::zpz<211>; using type =
04965
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<30>, ZPZV<61>, ZPZV<148>, ZPZV<148>, ZPZV<87>, ZPZV<125>,
                          ZPZV<2»; }; // NOLINT</pre>
04966
                                         template<> struct ConwayPolynomial<211, 11> { using ZPZ = aerobus::zpz<211>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<7>, ZPZV<209»; };</pre>
                                                                                                                            // NOLINT
                                          template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type
04967
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<50, ZPZV<145>, ZPZV<126>, ZPZV<184>,
                          ZPZV<84>, ZPZV<27>, ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<211, 13> { using ZPZ = aerobus::zpz<211>; using type =
04968
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          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<209»; }; // NOLINT
template<> struct ConwayPolynomial<211, 19> { using ZPZ = aerobus::zpz<211>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04970
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<209»; }; //</pre>
                         NOLINT
 04971
                                          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 =
04972
                        POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; }; // NOLINT
                                         template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
 04973
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
 04974
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<3»; }; // NOLINT
                                         template<> struct ConwayPolynomial<223, 5> { using ZPZ = aerobus::zpz<223>; using type =
 04975
                       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 =
 04976
                         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 =
 04977
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
 04978
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<3»; }; //
                         NOLINT
04979
                                          template<> struct ConwayPolynomial<223, 9> { using ZPZ = aerobus::zpz<223>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<164>, ZPZV<64>, ZPZV<220»;
                          }; // NOLINT
                         template<> struct ConwayPolynomial<223, 10> { using ZPZ = aerobus::zpz<223>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<118>, ZPZV<177>, ZPZV<87>, ZPZV<99>, ZPZV<62>,
 04980
                          ZPZV<3»: }: // NOLINT
                                         template<> struct ConwayPolynomial<223, 11> { using ZPZ = aerobus::zpz<223>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<8>, ZPZV<220»; }; // NOLINT</pre>
04982
                                       template<> struct ConwayPolynomial<223, 12> { using ZPZ = aerobus::zpz<223>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64>, ZPZV<94>, ZPZV<11>, ZPZV<164>,
                          ZPZV<151>, ZPZV<213>, ZPZV<3»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<223, 13> { using ZPZ = aerobus::zpz<223>; using type =
04983
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<220»; }; // NOLINT</pre>
04984
                                         template<> struct ConwayPolynomial<223,
                                                                                                                                                                                                               17> { using ZPZ = aerobus::zpz<223>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<20»; }; // NOLINT
template<> struct ConwayPolynomial<223, 19> { using ZPZ = aerobus::zpz<223>; using type :
04985
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7</pre>
                          NOLINT
 04986
                                         template<> struct ConwayPolynomial<227, 1> { using ZPZ = aerobus::zpz<227>; using type =
                       POLYV<ZPZV<1>, ZPZV<225»; }; // NOLINT
                                         template<> struct ConwayPolynomial<227, 2> { using ZPZ = aerobus::zpz<227>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2»; };
                                     template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<25»; }; // NOLINT template<> struct ConwayPolynomial<227, 4> { using ZPZ = aerobus::zpz<227>; using type =
 04989
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2»; ); // NOLINT template<> struct ConwayPolynomial<227, 5> { using ZPZ = aerobus::zpz<227>; using type =
04990
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<225»; }; // NOLINT
                                      template<> struct ConwayPolynomial<227, 6> { using ZPZ = aerobus::zpz<227>; using type =
 04991
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1A, ZPZV<24>, ZPZV<135>, ZPZV<2w; }; // NOLINT template<> struct ConwayPolynomial<227, 7> { using ZPZ = aerobus::zpz<227>; using type =
 04992
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25, ZPZV<18>, ZPZV<225»; }; // NOLINT template<> struct ConwayPolynomial<227, 8> { using ZPZ = aerobus::zpz<227>; using type =
04993
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<151>, ZPZV<176>, ZPZV<106>, ZPZV<2»; }; //
 04994
                                  template<> struct ConwayPolynomial<227, 9> { using ZPZ = aerobus::zpz<227>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>;
                      }; // NOLINT
04995
                                     template<> struct ConwayPolynomial<227, 10> { using ZPZ = aerobus::zpz<227>; using type :
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>, ZPZV<12>, ZPZV<12>, ZPZV<93>, ZPZV<77>,
                      ZPZV<2»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<2>, ZPZV<225»; }; // NOLINT</pre>
                      template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<123>, ZPZV<99>, ZPZV<160>, ZPZV<96>, ZPZV<127>, ZPZV<142>, ZPZV<94>, ZPZV<2»; }; // NOLINT
04997
                                   template<> struct ConwayPolynomial<227, 13> { using ZPZ = aerobus::zpz<227>; using type =
04998
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   V<0>, ZPZV<0>, ZPZV<2>, ZPZV<225»; }; // NOLINT template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type =
                      ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<225»; };</pre>
04999
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<225»; };</pre>
                                                                                                                                                                                                                                                                                                                    // NOLINT
                                     template<> struct ConwayPolynomial<227, 19> { using ZPZ = aerobus::zpz<227>; using type =
05000
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<225»; }; //</pre>
                      NOLINT
05001
                                     template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<223»; }; // NOLINT
 05002
                                      template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»; }; // NOLINT
05003
                                   template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; }; // NOLINT template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
 05004
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<6»; };
                                                                                                                                                                                                                                     // NOLINT
                                      template<> struct ConwayPolynomial<229, 5> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<23»; }; // NOLINT
 05006
                                  template<> struct ConwayPolynomial<229, 6> { using ZPZ = aerobus::zpz<229>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<160>, ZPZV<186>, ZPZV<6»; }; // NOLINT
05007
                                    template<> struct ConwayPolynomial<229, 7> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<23»; }; // NOLINT
                                     template<> struct ConwayPolynomial<229, 8> { using ZPZ = aerobus::zpz<229>; using type
 05008
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<193>, ZPZV<62>, ZPZV<205>, ZPZV<66; };
                      NOLINT
05009
                                    template<> struct ConwayPolynomial<229, 9> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<117>, ZPZV<50>, ZPZV<223»;
                      }; // NOLINT
                                      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</pre>
05011
                                      template<> struct ConwayPolynomial<229, 11> { using ZPZ = aerobus::zpz<229>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<229, 12> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<131>, ZPZV<140>, ZPZV<25>, ZPZV<6>, ZPZV<172>, ZPZV<9>, ZPZV<445>, ZPZV<6»; }; // NOLINT
05013
                                  template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type =
                       \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                      ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<223»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<23»; };</pre>
05015
                                  template<> struct ConwayPolynomial<229, 19> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      NOLINT
                                     template<> struct ConwayPolynomial<233, 1> { using ZPZ = aerobus::zpz<233>; using type =
                      POLYV<ZPZV<1>, ZPZV<230»; }; // NOLINT
 05017
                                    template<> struct ConwayPolynomial<233, 2> { using ZPZ = aerobus::zpz<233>; using type =
                     POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
                                    template<> struct ConwayPolynomial2233, 3> { using ZPZ = aerobus::zpz<233>; using type =
 05018
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<3»; }; // NOLINT
 05020
                                   template<> struct ConwayPolynomial<233, 5> { using ZPZ = aerobus::zpz<233>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<230»; }; // NOLINT
                      template<> struct ConwayPolynomial<233, 6> { using ZPZ = aerobus::zpz<233>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<122>, ZPZV<215>, ZPZV<32>, ZPZV<3»; }; // NOLINT</pre>
 05021
```

```
05022
                                        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<0>, ZPZV<4>, ZPZV<4>, ZPZV<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 8> { using ZPZ = aerobus::zpz<233>; using type =
05023
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<202>, ZPZV<135>, ZPZV<181>, ZPZV<3»; }; //
                        NOLINT
                                       template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type =
05024
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<7>, ZPZV<56>, ZPZV<146>, ZPZV<230»;
                        }; // NOLINT
                                       template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type =
05025
                        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 :
05026
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05027
                                     template<> struct ConwayPolynomial<233, 12> { using ZPZ = aerobus::zpz<233>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<21>, ZPZV<114>, ZPZV<114>, ZPZV<31>, ZPZV<19>, ZPZV<216>, ZPZV<20>, ZPZV<3»; }; // NOLINT
05028
                                      template<> struct ConwayPolynomial<233, 13> { using ZPZ = aerobus::zpz<233>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<230»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<233, 17> { using ZPZ = aerobus::zpz<233>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230»; }; // NOLINT
    template<> struct ConwayPolynomial<233, 19> { using ZPZ = aerobus::zpz<233>; using type =
POLYV<ZPZV<1>, ZPZV<0>, 
05030
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<23o»; }; //</pre>
05031
                                       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 =
05032
                       POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7»; }; // NOLINT
05033
                                        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 =
05034
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<239, 5> { using ZPZ = aerobus::zpz<239>; using type =
05035
                       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<232»; }; // NOLINT template<> struct ConwayPolynomial<239, 6> { using ZPZ = aerobus::zpz<239>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<237>, ZPZV<60>, ZPZV<200>, ZPZV<7»; }; // NOLINT
                                      template<> struct ConwayPolynomial<239, 7> { using ZPZ = aerobus::zpz<239>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<232»; };
                       template<> struct ConwayPolynomial<239, 8> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<202>, ZPZV<54>, ZPZV<7»; };</pre>
05038
                                       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»; };
05040
                                      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</pre>
                                        template<> struct ConwayPolynomial<239, 11> { using ZPZ = aerobus::zpz<239>; using type
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<8>, ZPZV<232»; }; // NOLINT</pre>
05042
                                       template<> struct ConwayPolynomial<239, 12> { using ZPZ = aerobus::zpz<239>; using type =
                       template<> struct ConwayPolynomial<239, 12> { using ZPZ = aerobus::zpz<239; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<235>, ZPZV<14>, ZPZV<113>, ZPZV<182>, ZPZV<101>, ZPZV<81>, ZPZV<216>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<239, 13> { using ZPZ = aerobus::zpz<239>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                          // NOLINT
                        ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; };</pre>
05044
                                        template<> struct ConwayPolynomial<239, 17> { using ZPZ = aerobus::zpz<239>; using type
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
05045
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<232»; }; //</pre>
                        NOLINT
05046
                                       template<> struct ConwayPolynomial<241, 1> { using ZPZ = aerobus::zpz<241>; using type =
                       POLYV<ZPZV<1>, ZPZV<234»; }; // NOLINT
                                       template<> struct ConwayPolynomial<241, 2> { using ZPZ = aerobus::zpz<241>; using type =
05047
                       POLYV<ZPZV<1>, ZPZV<238>, ZPZV<7»; }; // NOLINT
                                       template<> struct ConwayPolynomial<241, 3> { using ZPZ = aerobus::zpz<241>; using type =
05048
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<234»; }; // NOLINT template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
05049
                       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<14>, ZPZY<152>, ZPZY<7; }; // NOLINT
template<> struct ConwayPolynomial<241, 5> { using ZPZ = aerobus::zpz<241>; using type =
05050
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<234»; }; // NOLINT
                                       template<> struct ConwayPolynomial<241, 6> { using ZPZ = aerobus::zpz<241>; using type =
05051
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<6>, ZPZV<5>, ZPZV<7»; }; // NOLINT
05052
                                       template<> struct ConwayPolynomial<241, 7> { using ZPZ = aerobus::zpz<241>; using type =
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2, ZPZV<24*; ; / NOLINT template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
05053
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<113>, ZPZV<212>, ZPZV<153>, ZPZV<15»; }; //
05054
                                      template<> struct ConwayPolynomial<241, 9> { using ZPZ = aerobus::zpz<241>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<236>, ZPZV<125>, ZPZV<234»;
                        }; // NOLINT
05055
                                     template<> struct ConwayPolynomial<241, 10> { using ZPZ = aerobus::zpz<241>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<29>, ZPZV<27>, ZPZV<145>, ZPZV<208>, ZPZV<55>,
                        ZPZV<7»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<3>, ZPZV<234»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<241, 12> { using ZPZ = aerobus::zpz<241>; using type =
05057
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, 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 =
05058
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      template<> struct ConwayPolynomial<241, 17> { using ZPZ = aerobus::zpz<241>; using type
05059
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<234»; }; // NOLINT</pre>
05060
                                     template<> struct ConwayPolynomial<241, 19> { using ZPZ = aerobus::zpz<241>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                                                                                                                                                                                                                                                                                                                                                                7.P7.V<0>.
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234»; }; //</pre>
                        NOLINT
                                        template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
                       POLYV<ZPZV<1>, ZPZV<245»; }; // NOLINT
                                        template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
                       POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; }; // NOLINT
                                      template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
05063
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type =
05064
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<200>, ZPZV<6»; }; // NOLINT
                                      template<> struct ConwayPolynomial<251, 5> { using ZPZ = aerobus::zpz<251>; using type =
05065
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT
05066
                                        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 template<> struct ConwayPolynomial<251, 7> { using ZPZ = aerobus::zpz<251>; using type = aerobus::zpz<
05067
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<245»; };
                                        template<> struct ConwayPolynomial<251, 8> { using ZPZ = aerobus::zpz<251>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<142>, ZPZV<215>, ZPZV<173>, ZPZV<6»; }; //
                        NOLINT
                                       template<> struct ConwayPolynomial<251, 9> { using ZPZ = aerobus::zpz<251>; using type =
05069
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<48>, ZPZV<186>, ZPZV<106>, ZPZV<24»;
05070
                                         template<> struct ConwayPolynomial<251, 10> { using ZPZ = aerobus::zpz<251>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<138>, ZPZV<110>, ZPZV<45>, ZPZV<34>, ZPZV<149>, ZPZV<6»; }; // NOLINT
                                      template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type =
0.5071
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<26>, ZPZV<245»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<19>, ZPZV<53>, ZPZV<20>, ZPZV<20>, ZPZV<15>,
                        \text{ZPZV} < 201 >, \text{ZPZV} < 232 >, \text{ZPZV} < 6 >; }; // NOLINT
                                        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>
                                        template<> struct ConwayPolynomial<251,
                                                                                                                                                                                                      17> { using ZPZ = aerobus::zpz<251>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<245»; };</pre>
                                                                                                                                                                                                                                                                                                                                           // NOLINT
                       template<> struct ConwayPolynomial<251, 19> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05075
                         ZPZV<0>, ZPZV<0>
                                      template<> struct ConwayPolynomial<257, 1> { using ZPZ = aerobus::zpz<257>; using type =
                       POLYV<ZPZV<1>, ZPZV<254»; }; // NOLINT
05077
                                        template<> struct ConwayPolynomial<257, 2> { using ZPZ = aerobus::zpz<257>; using type =
                       POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3»; }; // NOLINT
                                      template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
05078
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<254»; }; // NOLINT
                                        template<> struct ConwayPolynomial<257, 4> { using ZPZ = aerobus::zpz<257>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<187>, ZPZV<3»; }; // NOLINT
05080
                                      template<> struct ConwayPolynomial<257, 5> { using ZPZ = aerobus::zpz<257>; using type =
                       template<> struct ConwayPolynomial<257, 6> { using ZPZ = aerobus::zpz<257>; using type =
05081
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<62>, ZPZV<18>, ZPZV<18>, ZPZV<138>, ZPZV<3»; }; // NOLINT
                                        template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type
05082
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<254»; }; // NOLI template<> struct ConwayPolynomial<257, 8> { using ZPZ = aerobus::zpz<257>; using type =
05083
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<179>, ZPZV<140>, ZPZV<162>, ZPZV<3»; }; //
                        NOLINT
                                       template<> struct ConwayPolynomial<257, 9> { using ZPZ = aerobus::zpz<257>; using type
05084
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<50>, ZPZV<254»;
                        }; // NOLINT
05085
                                        template<> struct ConwayPolynomial<257, 10> { using ZPZ = aerobus::zpz<257>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<97>, ZPZV<12>, ZPZV<225>, ZPZV<180>, ZPZV<20>,
                        ZPZV<3»: }: // NOLINT
                                        template<> struct ConwayPolynomial<257, 11> { using ZPZ = aerobus::zpz<257>; using type :
05086
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                        ZPZV<40>, ZPZV<254»; }; // NOLINT</pre>
05087
                                     template<> struct ConwayPolynomial<257, 12> { using ZPZ = aerobus::zpz<257>; using type =
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<13>, ZPZV<225>, ZPZV<215>, ZPZV<173>, ZPZV<249>, ZPZV<148>, ZPZV<20>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<257, 13> { using ZPZ = aerobus::zpz<257>; using type =
05088
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<254»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<257, 17> { using ZPZ = aerobus::zpz<257>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<254»; }; // NOLINT
template<> struct ConwayPolynomial<257, 19> { using ZPZ = aerobus::zpz<257>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<254»; }; //</pre>
05091
                                template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
                   POLYV<ZPZV<1>, ZPZV<258»; }; // NOLINT
                               05092
                   POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; }; // NOLINT
                                template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; }; // NOLINT
                               template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
 05094
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; }; // NOLINT
                                template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
05095
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<258»; }; // NOLINT
                                template<> struct ConwayPolynomial<263, 6> { using ZPZ = aerobus::zpz<263>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<22>, ZPZV<250>, ZPZV<225>, ZPZV<25»; }; // NOLINT template<> struct ConwayPolynomial<263, 7> { using ZPZ = aerobus::zpz<263>; using type =
 05097
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<258»; }; // NOLINT template<> struct ConwayPolynomial<263, 8> { using ZPZ = aerobus::zpz<263>; using type =
 05098
                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<227>, ZPZV<170>, ZPZV<7>, ZPZV<5»; };
                               template<> struct ConwayPolynomial<263, 9> { using ZPZ = aerobus::zpz<263>; using type =
05099
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<261>, ZPZV<29>, ZPZV<258»;
                   }; // NOLINT
05100
                                template<> struct ConwayPolynomial<263, 10> { using ZPZ = aerobus::zpz<263>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<245>, ZPZV<231>, ZPZV<198>, ZPZV<145>,
                   ZPZV<119>, ZPZV<5»; };</pre>
                                                                                              // NOLINT
                                template<> struct ConwayPolynomial<263, 11> { using ZPZ = aerobus::zpz<263>; using type =
05101
                   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 =
05102
                   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 template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
05103
                   POLYV<ZPZV<1>, ZPZV<267»; }; // NOLINT
05104
                              template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
                  POLYV<ZPZV<1>, ZPZV<268>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
 05105
                  POLYV<ZPZV<1>, ZPZV<2>, ZPZV<9>, ZPZV<267»; }; // NOLINT template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2»; }; // NOLINT
 05107
                              template<> struct ConwayPolynomial<269, 5> { using ZPZ = aerobus::zpz<269>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<267»; }; // NOLINT
05108
                                template<> struct ConwayPolynomial<269, 6> { using ZPZ = aerobus::zpz<269>; using type =
                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<120>, ZPZV<101>, ZPZV<206>, ZPZV<2»; }; // NOLINT
05109
                               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
                              template<> struct ConwayPolynomial<269, 8> { using ZPZ = aerobus::zpz<269>; using type =
 0.5110
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<220>, ZPZV<131>, ZPZV<232>, ZPZV<23; }; //
                   NOLINT
05111
                                template<> struct ConwayPolynomial<269, 9> { using ZPZ = aerobus::zpz<269>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<214>, ZPZV<267>, ZPZV<267»;
                   }; // NOLINT
                   template<> struct ConwayPolynomial<269, 10> { using ZPZ = aerobus::zpz<269>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<264>, ZPZV<243>, ZPZV<186>, ZPZV<61>,
                   ZPZV<10>, ZPZV<2»; };</pre>
                                                                                           // NOLINT
                               template<> struct ConwayPolynomial<269, 11> { using ZPZ = aerobus::zpz<269>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   ZPZV<20>, ZPZV<267»; };</pre>
                                                                                              // NOLINT
                              template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type =
05114
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<165>, ZPZV<663>, ZPZV<215>,
                   ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type =
05115
                   POLYV<ZPZV<1>, ZPZV<265»; }; // NOLINT
 05116
                                template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
                   POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; }; // NOLINT
 05117
                              template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type =
 05118
                  POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<205>, ZPZV<6%; }; // NOLINT
template<> struct ConwayPolynomial<271, 5> { using ZPZ = aerobus::zpz<271>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT
 05120
                                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<65; }; // NOLINT template<> struct ConwayPolynomial<271, 7> { using ZPZ = aerobus::zpz<271>; using type :
05121
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT
                                template<> struct ConwayPolynomial<271, 8> { using ZPZ = aerobus::zpz<271>; using type =
                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<114>, ZPZV<69>, ZPZV<69»; };
                   NOLINT
05123
                              template<> struct ConwayPolynomial<271, 9> { using ZPZ = aerobus::zpz<271>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<266>, ZPZV<186>, ZPZV<265»;
                   }; // NOLINT
```

```
template<> struct ConwayPolynomial<271, 10> { using ZPZ = aerobus::zpz<271>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<133>, ZPZV<10>, ZPZV<256>, ZPZV<74>, ZPZV<126>, ZPZV<6»; }; // NOLINT
                        template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type =
05125
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<10>, ZPZV<265»; }; // NOLINT
template<> struct ConwayPolynomial<271, 12> { using ZPZ = aerobus::zpz<271>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162>, ZPZV<210>, ZPZV<216>, ZPZV<216-, ZPZV<216-,
                ZPZV<237>, ZPZV<256>, ZPZV<130>, ZPZV<6»; }; // NOLINT</pre>
05127
                         template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<272»; }; // NOLINT
                         template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
05128
               POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; }; // NOLINT
                          template<> struct ConwayPolynomial2277, 3> { using ZPZ = aerobus::zpz<277>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; }; // NOLINT

template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<5»; }; // NOLINT
05130
                         template<> struct ConwayPolynomial<277, 5> { using ZPZ = aerobus::zpz<277>; using type =
05131
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<272»; }; // NOLINT
                         template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type =
               POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<3>, ZPZY<3>, ZPZY<9>, ZPZY<118>, ZPZY<5>; }; // NOLINT template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type =
05133
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<277, 8> { using ZPZ = aerobus::zpz<277>; using type =
05134
               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 =
05135
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177>, ZPZV<110>, ZPZV<272»;
               }; // NOLINT
05136
                         template<> struct ConwayPolynomial<277, 10> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<206>, ZPZV<253>, ZPZV<237>, ZPZV<241>,
               ZPZV<260>, ZPZV<5»; };</pre>
                                                                            // NOLINT
                          template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type =
05137
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                ZPZV<5>, ZPZV<272»; }; // NOLINT</pre>
05138
                          template<> struct ConwayPolynomial<277, 12> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<183>, ZPZV<218>, ZPZV<240>, ZPZV<40>, ZPZV<40>, ZPZV<180>, ZPZV<115>, ZPZV<202>, ZPZV<5»; }; // NOLINT
05139
                          template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<278»; }; // NOLINT
05140
                         template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
0.5141
               POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<278; }; // NOLINT
template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3»; }; // NOLINT
05143
                        template<> struct ConwayPolynomial<281, 5> { using ZPZ = aerobus::zpz<281>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<278»; }; // NOLINT
05144
                         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
05145
                         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<19>, ZPZV<278»; }; //
                         template<> struct ConwayPolynomial<281, 8> { using ZPZ = aerobus::zpz<281>; using type =
05146
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195>, ZPZV<279>, ZPZV<140>, ZPZV<3»; }; //
               NOLINT
05147
                         template<> struct ConwayPolynomial<281, 9> { using ZPZ = aerobus::zpz<281>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<148>, ZPZV<70>, ZPZV<278»;
               }; // NOLINT
               template<> struct ConwayPolynomial<281, 10> { using ZPZ = aerobus::zpz<281>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<145>, ZPZV<13>, ZPZV<138>,
               ZPZV<191>, ZPZV<3»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<281, 11> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<36>, ZPZV<278»; };</pre>
                                                                            // NOLINT
                        template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type =
05150
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<202>, ZPZV<68>, ZPZV<103>, ZPZV<116>,
               ZPZV<58>, ZPZV<28>, ZPZV<191>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
05151
               POLYV<ZPZV<1>, ZPZV<280»; }; // NOLINT
05152
                          template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
               POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3»; }; // NOLINT
05153
                        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 =
05154
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
05156
                         template<> struct ConwayPolynomial<283, 6> { using ZPZ = aerobus::zpz<283>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<68>, ZPZV<73>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type =
05157
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<280»; }; // NOLINT
                         template<> struct ConwayPolynomial<283, 8> { using ZPZ = aerobus::zpz<283>; using type =
05158
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<179>, ZPZV<32>, ZPZV<232>, ZPZV<23»; }; //
               NOLINT
05159
                        template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<65>, ZPZV<280»;
               }; // NOLINT
```

```
template<> struct ConwayPolynomial<283, 10> { using ZPZ = aerobus::zpz<283>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<271>, ZPZV<185>, ZPZV<68>, ZPZV<100>, ZPZV<219>, ZPZV<3»; }; // NOLINT
               \label{eq:convergence} template<> struct ConwayPolynomial<283, 11> \{ using ZPZ = aerobus::zpz<283>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
05161
               ZPZV<4>, ZPZV<280»; };</pre>
                                                                            // NOLINT
                         template<> struct ConwayPolynomial<283, 12> { using ZPZ = aerobus::zpz<283>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<8>, ZPZV<8>, ZPZV<86>, ZPZV<26>, ZPZV<49>,
               ZPZV<14>, ZPZV<56>, ZPZV<3»; }; // NOLINT</pre>
05163
                         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 =
05164
               POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; }; // NOLINT
                          template<> struct ConwayPolynomial<293, 3> { using ZPZ = aerobus::zpz<293>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT template<> struct ConwayPolynomial<293, 4> { using ZPZ = aerobus::zpz<293>; using type =
05166
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
05167
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
05168
                         template<> struct ConwayPolynomial<293, 6> { using ZPZ = aerobus::zpz<293>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<210>, ZPZV<260>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<293, 7> { using ZPZ = aerobus::zpz<293>; using type =
05169
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<291»; }; // NOLINT template<> struct ConwayPolynomial<293, 8> { using ZPZ = aerobus::zpz<293>; using type =
0.5170
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<239>, ZPZV<29>; }; //
                        template<> struct ConwayPolynomial<293, 9> { using ZPZ = aerobus::zpz<293>; using type
05171
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<209, ZPZV<209, ZPZV<209, ZPZV<208, ZPZV<209, ZPZV<20
               }; // NOLINT
                         template<> struct ConwayPolynomial<293, 10> { using ZPZ = aerobus::zpz<293>; using type =
05172
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<186>, ZPZV<28>, ZPZV<46>, ZPZV<184>, ZPZV<24>,
               ZPZV<2»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<293, 11> { using ZPZ = aerobus::zpz<293>; using type =
05173
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
               ZPZV<3>, ZPZV<291»; }; // NOLINT</pre>
05174
                         template<> struct ConwayPolynomial<293, 12> { using ZPZ = aerobus::zpz<293>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<159>, ZPZV<210>, ZPZV<125>, ZPZV<212>, ZPZV<167>, ZPZV<144>, ZPZV<157>, ZPZV<22; }; // NOLINT
05175
                          template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
               POLYV<ZPZV<1>, ZPZV<302»; }; // NOLINT
05176
                        template<> struct ConwayPolynomial<307, 2> { using ZPZ = aerobus::zpz<307>; using type =
              POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
05177
              POLYV<ZPZV<1>, ZPZV<3>, ZPZV<7>, ZPZV<302»; }; // NOLINT template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5»; }; // NOLINT
05179
                        template<> struct ConwayPolynomial<307, 5> { using ZPZ = aerobus::zpz<307>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<302»; }; // NOLINT
                         template<> struct ConwayPolynomial<307, 6> { using ZPZ = aerobus::zpz<307>; using type =
05180
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<213>, ZPZV<172>, ZPZV<61>, ZPZV<5»; }; // NOLINT
05181
                         template<> struct ConwayPolynomial<307,
                                                                                                                              7> { using ZPZ = aerobus::zpz<307>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                        template<> struct ConwayPolynomial<307, 8> { using ZPZ = aerobus::zpz<307>; using type =
05182
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<232>, ZPZV<131>, ZPZV<5»; }; //
               NOLINT
               template<> struct ConwayPolynomial<307, 9> { using ZPZ = aerobus::zpz<307>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<165>, ZPZV<165>, ZPZV<70>, ZPZV<302»;
05183
               }; // NOLINT
                         template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
05184
               POLYV<ZPZV<1>, ZPZV<294»; }; // NOLINT
                        template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
05185
               POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; }; // NOLINT
                         template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
05186
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; }; // NOLINT template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
05187
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; }; // NOLINT
                         template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
05188
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294»; }; // NOLINT template<> struct ConwayPolynomial<311, 6> { using ZPZ = aerobus::zpz<311>; using type =
05189
              POLYV<2PZV<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 =
05190
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<294»; };
               template<> struct ConwayPolynomial<311, 8> { using ZPZ = aerobus::zpz<311>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<162>, ZPZV<118>, ZPZV<2>, ZPZV<17»; }; //</pre>
05191
               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<0>, ZPZV<1>, ZPZV<287>, ZPZV<287>, ZPZV<24>,
               }; // NOLINT
05193
                         template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
               POLYV<ZPZV<1>, ZPZV<303»; }; // NOLINT
                         template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
05194
               POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; }; // NOLINT
                         template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; };
                                                                                                                                         // NOLINT
                       template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
05196
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<23>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
05197
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<303»; };
               template<> struct ConwayPolynomial<313, 6> { using ZPZ = aerobus::zpz<313>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<196>, ZPZV<213>, ZPZV<253>, ZPZV<10»; }; // NOLINT
05199
               template<> struct ConwayPolynomial<313, 7> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<303»; }; // NOLINT template<> struct ConwayPolynomial<313, 8> { using ZPZ = aerobus::zpz<313>; using type =
05200
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<306>, ZPZV<99>, ZPZV<106>, ZPZV<100>; }; //
05201
               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<8>, ZPZV<267>, ZPZV<300>, ZPZV<303»;
         }; // NOLINT
               template<> struct ConwayPolynomial<317, 1> { using ZPZ = aerobus::zpz<317>; using type =
05202
         POLYV<ZPZV<1>, ZPZV<315»; // NOLINT
               template<> struct ConwayPolynomial<317, 2> { using ZPZ = aerobus::zpz<317>; using type =
         POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; }; // NOLINT
05204
               template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<315»; }; // NOLINT template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
05205
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2»; };
                                                                                                 // NOLINT
               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
05207
               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 template<> struct ConwayPolynomial<317, 7> { using ZPZ = aerobus::zpz<317>; using type = aerobus::zpz<31
05208
        POLYV-ZPZV-1>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-315s; }; // NoLII template<> struct ConwayPolynomial<317, 8> { using ZPZ = aerobus::zpz<317>; using type :
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<207>, ZPZV<85>, ZPZV<31>, ZPZV<2»; };
         NOLINT
         template<> struct ConwayPolynomial<317, 9> { using ZPZ = aerobus::zpz<317>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<284>, ZPZV<296>, ZPZV<315»;</pre>
05210
         }; // NOLINT
05211
                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 =
05212
         POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; }; // NOLINT
05213
               template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<328»; }; // NOLINT template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<290>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<331, 5> { using ZPZ = aerobus::zpz<331>; using type =
05215
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<328»; }; // NOLINT
               template<> struct ConwayPolynomial<331, 6> { using ZPZ = aerobus::zpz<331>; using type =
05216
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<205>, ZPZV<15>, 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*; }; //
05218
               template<> struct ConwayPolynomial<331, 8> { using ZPZ = aerobus::zpz<331>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<249>, ZPZV<308>, ZPZV<78>, ZPZV<3»; };
         NOLINT
05219
               template<> struct ConwayPolynomial<331, 9> { using ZPZ = aerobus::zpz<331>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<194>, ZPZV<210>, ZPZV<328»;
         }; // NOLINT
  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 =
05221
         POLYV<ZPZV<1>, ZPZV<332>, ZPZV<10»; }; // NOLINT
               template<> struct ConwayPolynomial<337, 3> { using ZPZ = aerobus::zpz<337>; using type =
05222
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT
               template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
05223
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10»; }; // NOLINT
               template<> struct ConwayPolynomial<337, 5> { using ZPZ = aerobus::zpz<337>; using type =
05224
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<32>, ZPZV<327>; }; // NOLINT template<> struct ConwayPolynomial<337, 6> { using ZPZ = aerobus::zpz<337>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<216>, ZPZV<107>, ZPZV<109>, ZPZV<10»; }; // NOLINT
05225
               template<> struct ConwayPolynomial<337, 7> { using ZPZ = aerobus::zpz<337>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<327»; };
05227
              template<> struct ConwayPolynomial<337, 8> { using ZPZ = aerobus::zpz<337>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<331>, ZPZV<246>, ZPZV<251>, ZPZV<10»; }; //
         NOLINT
               template<> struct ConwayPolynomial<337, 9> { using ZPZ = aerobus::zpz<337>; using type =
05228
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<148>, ZPZV<98>, ZPZV<327»;
         }; // NOLINT
05229
               template<> struct ConwayPolynomial<347, 1> { using ZPZ = aerobus::zpz<347>; using type =
         POLYV<ZPZV<1>, ZPZV<345»; }; // NOLINT
               template<> struct ConwayPolynomial<347, 2> { using ZPZ = aerobus::zpz<347>; using type =
05230
         POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<9>, ZPZV<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
05232
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
05233
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<345»; }; // NOLINT
               template<> struct ConwayPolynomial<347, 6> { using ZPZ = aerobus::zpz<347>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<343>, ZPZV<26>, ZPZV<56>, ZPZV<28; }; // NOLINT template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
05235
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<213>, ZPZV<213>, ZPZV<2117>, ZPZV<2»; }; //
05236
```

```
NOLINT
       template<> struct ConwayPolynomial<347, 9> { using ZPZ = aerobus::zpz<347>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<25>, ZPZV<252>, ZPZV<345»;
       }; // NOLINT
05238
            template<> struct ConwayPolynomial<349, 1> { using ZPZ = aerobus::zpz<349>; using type =
       POLYV<ZPZV<1>, ZPZV<347»; }; // NOLINT
            template<> struct ConwayPolynomial<349, 2> { using ZPZ = aerobus::zpz<349>; using type =
       POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; }; // NOLINT
05240
           template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; }; // NOLINT template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
05241
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<22; ); // NOLINT template<> struct ConwayPolynomial<349, 5> { using ZPZ = aerobus::zpz<349>; using type =
05242
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<347»; }; // NOLINT
05243
           template<> struct ConwayPolynomial<349, 6> { using ZPZ = aerobus::zpz<349>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<135>, ZPZV<177>, ZPZV<316>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<349, 7> { using ZPZ = aerobus::zpz<349>; using type =
05244
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<347»; }; // NOLINT
            template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<308>, ZPZV<328>, ZPZV<268>, ZPZV<2*; }; //
05246
            template<> struct ConwayPolynomial<349, 9> { using ZPZ = aerobus::zpz<349>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<290>, ZPZV<130>, ZPZV<347»;
       }; // NOLINT
05247
            template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<ZPZV<1>, ZPZV<350»; }; // NOLINT
           template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
05248
       POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3»; }; // NOLINT
05249
            template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<350»; }; // NOLINT
template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<3»; }; // NOLINT
05250
            template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
05251
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<350»; }; // NOLINT
05252
            template<> struct ConwayPolynomial<353, 6> { using ZPZ = aerobus::zpz<353>; using type =
       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
05253
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<350»; }; // NOLINT
            template<> struct ConwayPolynomial<353, 8> { using ZPZ = aerobus::zpz<353>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<182>, ZPZV<26>, ZPZV<37>, ZPZV<3»; };
05255
       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»;</pre>
       }; // NOLINT
    template<> struct ConwayPolynomial<359, 1> { using ZPZ = aerobus::zpz<359>; using type =
       POLYV<ZPZV<1>, ZPZV<352»; }; // NOLINT
05257
           template<> struct ConwayPolynomial<359, 2> { using ZPZ = aerobus::zpz<359>; using type =
       POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
05258
       POLYY<ZPZY<1>, ZPZY<3>, ZPZY<3>, ZPZY<352,; }; // NOLINT template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
05259
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<229>, ZPZV<7»; }; // NOLINT
05260
            template<> struct ConwayPolynomial<359, 5> { using ZPZ = aerobus::zpz<359>; using type =
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; }; // NOLINT template<> struct ConwayPolynomial<359, 6> { using ZPZ = aerobus::zpz<359>; using type =
05261
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<309, ZPZV<327>, ZPZV<327>, ZPZV<327>; }; // NOLINT template<> struct ConwayPolynomial<359, 7> { using ZPZ = aerobus::zpz<359>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; };
           template<> struct ConwayPolynomial<359, 8> { using ZPZ = aerobus::zpz<359>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<301>, ZPZV<143>, ZPZV<271>, ZPZV<7»; }; //
       NOLINT
       template<> struct ConwayPolynomial<359, 9> { using ZPZ = aerobus::zpz<359>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<356>, ZPZV<356>, ZPZV<356>, ZPZV<352»;
05264
       }; // NOLINT
           template<> struct ConwayPolynomial<367, 1> { using ZPZ = aerobus::zpz<367>; using type =
05265
       POLYV<ZPZV<1>, ZPZV<361»; }; // NOLINT
05266
            template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
       POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6»; }; // NOLINT
            template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
05267
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<361»; }; // NOLINT template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type =
       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
05269
            template<> struct ConwayPolynomial<367, 6> { using ZPZ = aerobus::zpz<367>; using type =
05270
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<222>, ZPZV<321>, ZPZV<324>, ZPZV<6»; }; // NOLINI
           template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type
05271
       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 =
05272
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<335>, ZPZV<282>, ZPZV<50>, ZPZV<6»; };
       NOLINT
            template<> struct ConwayPolynomial<367, 9> { using ZPZ = aerobus::zpz<367>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<213>, ZPZV<268>, ZPZV<361»;
       }; // NOLINT
05274
           template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<371»; }; // NOLINT
           template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2»; };
           template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<371»; }; // NOLINT template<> struct ConwayPolynomial<373, 4> { using ZPZ = aerobus::zpz<373>; using type =
05277
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<373, 5> { using ZPZ = aerobus::zpz<373>; using type =
05278
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<371»; }; // NOLINT
05279
           template<> struct ConwayPolynomial<373, 6> { using ZPZ = aerobus::zpz<373>; using type =
      05280
          template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<371»; }; // NOLINT template<> struct ConwayPolynomial<373, 8> { using ZPZ = aerobus::zpz<373>; using type =
05281
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<203, ZPZV<219>, ZPZV<66>, ZPZV<2»; }; //
05282
          template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<238>, ZPZV<370>, ZPZV<371»;
      }; // NOLINT
           template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
05283
      POLYV<ZPZV<1>, ZPZV<377»; }; // NOLINT
           template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2»; }; // NOLINT
05285
           template<> struct ConwayPolynomial<379, 3> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377»; }; // NOLINT
template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327>, ZPZV<2»; }; // NOLINT
05286
           template<> struct ConwayPolynomial<379, 5> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<377»; }; // NOLINT
05288
           template<> struct ConwayPolynomial<379, 6> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<374>, ZPZV<364>, ZPZV<246>, ZPZV<2»; }; // NOLINT
05289
          template<> struct ConwayPolynomial<379, 7> { using ZPZ = aerobus::zpz<379>; using type =
      POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV×0>, ZPZV<377»; }; // NOLINT 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»; }; //
      template<> struct ConwayPolynomial<379, 9> { using ZPZ = aerobus::zpz<379>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<362>, ZPZV<369>, ZPZV<377»;</pre>
05291
      }; // NOLINT
           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 =
05293
      POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
05294
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
05295
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5»; }; // NOLINT
05296
           template<> struct ConwayPolynomial<383, 5> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
05297
           template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<8>, ZPZV<158>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<383, 7> { using ZPZ = aerobus::zpz<383>; using type =
05298
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<378»; };
           template<> struct ConwayPolynomial<383, 8> { using ZPZ = aerobus::zpz<383>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<281>, ZPZV<332>, ZPZV<296>, ZPZV<5»; }; //
05300
           template<> struct ConwayPolynomial<383, 9> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137>, ZPZV<76>, ZPZV<378»;
05301
          template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<387»; }; // NOLINT
05302
           template<> struct ConwayPolynomial<389, 2> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<379>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
05303
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387»; }; // NOLINT template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<266>, ZPZV<2»; }; // NOLINT
05305
           template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
      template<> struct ConwayPolynomial<389, 6> { using ZPZ = aerobus::zpz<389>; using type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<339>, ZPZV<255>, ZPZV<2»; }; // NOLINT
05306
           template<> struct ConwayPolynomial<389,
                                                         7> { using ZPZ = aerobus::zpz<389>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<387»; };
05308
          template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<351>, ZPZV<19>, ZPZV<290>, ZPZV<2»; }; //
      NOLINT
           template<> struct ConwayPolynomial<389, 9> { using ZPZ = aerobus::zpz<389>; using type =
05309
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<308>, ZPZV<387»;
05310
           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 =
05311
      POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<397, 3> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<392»; }; // NOLINT template<> struct ConwayPolynomial<397, 4> { using ZPZ = aerobus::zpz<397>; using type =
05313
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
05314
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392»; }; // NOLINT
```

```
05315
                    template<> struct ConwayPolynomial<397, 6> { using ZPZ = aerobus::zpz<397>; using type =
            POLYY<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 =
            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 =
05317
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<375>, ZPZV<255>, ZPZV<203>, ZPZV<5»; }; //
                    template<> struct ConwayPolynomial<397, 9> { using ZPZ = aerobus::zpz<397>; using type
05318
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<166>, ZPZV<166>, ZPZV<252>, ZPZV<392»;
            }; // NOLINT
05319
                    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 =
05320
            POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3»; }; // NOLINT
05321
                   template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<398»; ); // NOLINT
template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<401, 5> { using ZPZ = aerobus::zpz<401>; using type =
05322
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<398»; }; // NOLINT
                    template<> struct ConwayPolynomial<401, 6> { using ZPZ = aerobus::zpz<401>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<115>, ZPZV<81>, ZPZV<51>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<401, 7> { using ZPZ = aerobus::zpz<401>; using type =
05325
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05326
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3», ZPZV<113>, ZPZV<164>, ZPZV<3»; }; //
            template<> struct ConwayPolynomial<401, 9> { using ZPZ = aerobus::zpz<401>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<198>, ZPZV<398»;</pre>
05327
            }; // NOLINT
                    template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
05328
            POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
                    template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
            POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT
                    template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
05330
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388»; }; // NOLINT template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<407>, ZPZV<21»; }; // NOLINT
05331
05332
                     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 =
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 =
05333
05334
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                    template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<256>, ZPZV<69>, ZPZV<396>, ZPZV<31»; }; //
            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<318>, ZPZV<318>, ZPZV<318>,
05336
            }; // NOLINT
05337
                     template<> struct ConwayPolynomial<419, 1> { using ZPZ = aerobus::zpz<419>; using type =
            POLYV<ZPZV<1>, ZPZV<417»; }; // NOLINT
05338
                    template<> struct ConwayPolynomial<419, 2> { using ZPZ = aerobus::zpz<419>; using type =
            POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
05339
            POLYY<ZPZY<1>, ZPZV<0>, ZPZV<11>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<373>, ZPZV<2»; }; // NOLINT
                    template<> struct ConwayPolynomial<419, 5> { using ZPZ = aerobus::zpz<419>; using type =
05341
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; // NOLINT
                   template<> struct ConwayPolynomial<419, 6> { using ZPZ = aerobus::zpz<419>; using type =
05342
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<411>, ZPZV<257>, ZPZV<257>, ZPZV<249; ); // NOLINT template<> struct ConwayPolynomial<419, 7> { using ZPZ = aerobus::zpz<419>; using type
05343
            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4+, ZPZV<417»; }; //
05344
                    template<> struct ConwayPolynomial<419, 8> { using ZPZ = aerobus::zpz<419>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<234>, ZPZV<388>, ZPZV<151>, ZPZV<2»; }; //
            NOLINT
05345
                   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<0>, ZPZV<0>, ZPZV<386>, ZPZV<417»;
            }; // NOLINT
                    template<> struct ConwayPolynomial<421, 1> { using ZPZ = aerobus::zpz<421>; using type =
05346
            POLYV<ZPZV<1>, ZPZV<419»; }; // NOLINT
                    template<> struct ConwayPolynomial<421, 2> { using ZPZ = aerobus::zpz<421>; using type =
05347
            POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419»; }; // NOLINT
05348
                    template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
05349
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<421, 5> { using ZPZ = aerobus::zpz<421>; using type =
05350
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<419»; }; // NOLINT template<> struct ConwayPolynomial<421, 6> { using ZPZ = aerobus::zpz<421>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<111>, ZPZV<342>, ZPZV<41>, ZPZV<2»; }; // NOLINT
05351
                    template<> struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<21>, ZPZV<419»; };
05353
                   template<> struct ConwayPolynomial<421, 8> { using ZPZ = aerobus::zpz<421>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<389>, ZPZV<32>, ZPZV<77>, ZPZV<2»; }; //
            NOLTNT
```

```
template<> struct ConwayPolynomial<421, 9> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<145>, ZPZV<4145>, ZPZV<4145>, ZPZV<419»;
      }; // NOLINT
05355
          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 =
05356
      POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<431, 3> { using ZPZ = aerobus::zpz<431>; using type =
05357
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424%; }; // NOLINT template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
05358
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<431, 5> { using ZPZ = aerobus::zpz<431>; using type =
05359
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<431, 6> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<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 =
05361
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<424»; }; // NOLINT
          template<> struct ConwayPolynomial<431, 8> { using ZPZ = aerobus::zpz<431>; using type =
05362
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<243>, ZPZV<286>, ZPZV<115>, ZPZV<7»; }; //
      template<> struct ConwayPolynomial<431, 9> { using ZPZ = aerobus::zpz<431>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<71>, ZPZV<329>, ZPZV<424*;
05363
      }; // NOLINT
           template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
05364
      POLYV<ZPZV<1>, ZPZV<428»; }; // NOLINT
           template<> struct ConwayPolynomial<433, 2> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<433, 3> { using ZPZ = aerobus::zpz<433>; using type =
05366
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428»; }; // NOLINT template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
05367
      POLYVCZPZV<1>, ZPZV<6>, ZPZV<60, ZPZV<402>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<433, 5> { using ZPZ = aerobus::zpz<433>; using type =
05368
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<428»; }; // NOLINT
          template<> struct ConwayPolynomial<433, 6> { using ZPZ = aerobus::zpz<433>; using type =
05369
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<244>, ZPZV<353>, ZPZV<360>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<433, 7> { using ZPZ = aerobus::zpz<433>; using type =
05370
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<428»; }; // NOLINT
           template<> struct ConwayPolynomial<433, 8> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347>, ZPZV<32>, ZPZV<39>, ZPZV<5»; };
05372
          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<0>, ZPZV<2>, ZPZV<23>, ZPZV<45>, ZPZV<428»;
      }; // NOLINT
05373
           template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
05374
           template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; }; // NOLINT
          template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
05375
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
05376
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; };
                                                                      // NOLINT
           template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT
05378
           template<> struct ConwayPolynomial<439, 6> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<324>, ZPZV<190>, ZPZV<15»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type
05379
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; //
          template<> struct ConwayPolynomial<439, 8> { using ZPZ = aerobus::zpz<439>; using type =
05380
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<359>, ZPZV<296>, ZPZV<266>, ZPZV<266>, ZPZV<15»; }; //
      NOLINT
          template<> struct ConwayPolynomial<439, 9> { using ZPZ = aerobus::zpz<439>; using type =
05381
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<342>, ZPZV<254>, ZPZV<424»;
      }; // NOLINT
           template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<441»; }; // NOLINT
05383
          template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; }; // NOLINT
          template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
05384
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT
           template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
05385
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2»; }; // NOLINT
          template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<443>; using type =
05386
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT template<> struct ConwayPolynomial<443, 6> { using ZPZ = aerobus::zpz<443>; using type =
05387
      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
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<441»; };
05389
          template<> struct ConwayPolynomial<443, 8> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<217>, ZPZV<290>, ZPZV<2»; }; //
      NOLINT
           template<> struct ConwayPolynomial<443, 9> { using ZPZ = aerobus::zpz<443>; using type =
05390
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<125>, ZPZV<109>, ZPZV<441»;
      }; // NOLINT
05391
           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 =
05392
      POLYV<ZPZV<1>, ZPZV<444>, ZPZV<3»; }; // NOLINT
```

```
05393
               template<> struct ConwayPolynomial<449, 3> { using ZPZ = aerobus::zpz<449>; using type =
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<1>, ZPZV<446»; }; // NOLINT template<> struct ConwayPolynomial<449, 4> { using ZPZ = aerobus::zpz<449>; using type =
05394
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<449, 5> { using ZPZ = aerobus::zpz<449>; using type =
05395
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<446»; }; // NOLINT
              template<> struct ConwayPolynomial<449, 6> { using ZPZ = aerobus::zpz<449>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<293>, ZPZV<69>, ZPZV<3»; }; // NOLINI
              template<> struct ConwayPolynomial<449, 7> { using ZPZ = aerobus::zpz<449>; using type
05397
        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 =
05398
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361>, ZPZV<348>, ZPZV<124>, ZPZV<3*; }; //
              template<> struct ConwayPolynomial<449, 9> { using ZPZ = aerobus::zpz<449>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<226>, ZPZV<9>, ZPZV<4446»; };
              template<> struct ConwayPolynomial<457, 1> { using ZPZ = aerobus::zpz<457>; using type =
05400
        POLYV<ZPZV<1>, ZPZV<444»; }; // NOLINT
              template<> struct ConwayPolynomial<457, 2> { using ZPZ = aerobus::zpz<457>; using type =
        POLYV<ZPZV<1>, ZPZV<454>, ZPZV<13»; }; // NOLINT
               template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444*; }; // NOLINT template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
05403
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13»; }; // NOLINT
template<> struct ConwayPolynomial<457, 5> { using ZPZ = aerobus::zpz<457>; using type =
05404
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44+»; }; // NOLINT
              template<> struct ConwayPolynomial<457, 6> { using ZPZ = aerobus::zpz<457>; using type =
05405
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<205, ZPZV<389>, ZPZV<266>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<457, 7> { using ZPZ = aerobus::zpz<457>; using type
05406
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<444*; }; // NOLINT
              template<> struct ConwayPolynomial<457, 8> { using ZPZ = aerobus::zpz<457>; using type =
05407
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<4>>, ZPZV<296>, ZPZV<412>, ZPZV<13»; };
             template<> struct ConwayPolynomial<457, 9> { using ZPZ = aerobus::zpz<457>; using type =
05408
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<444*;
         }; // NOLINT
        template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type = POLYV<ZPZV<1>, ZPZV<459»; }; // NOLINT
05409
               template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2»; }; // NOLINT
05411
              template<> struct ConwayPolynomial<461, 3> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459»; }; // NOLINT template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2»; }; // NOLINT
05412
              template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<459»; }; // NOLINT
05414
              template<> struct ConwayPolynomial<461, 6> { using ZPZ = aerobus::zpz<461>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<439>, ZPZV<432>, ZPZV<329>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<461, 7> { using ZPZ = aerobus::zpz<461>; using type =
05415
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<459»; }; // NOLINT
              template<> struct ConwayPolynomial<461, 8> { using ZPZ = aerobus::zpz<461>; using type
05416
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<449>, ZPZV<321>, ZPZV<2»; }; //
        NOLINT
        \label{eq:convergence} $$ \text{template<> struct ConwayPolynomial<461, 9> { using ZPZ = aerobus::zpz<461>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<210>, ZPZV<210>, ZPZV<25>, ZPZV<210>, ZPZV<276>, ZPZV<459w; ZPZV<210>, ZPZV<210
05417
        }; // NOLINT
05418
               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 =
05419
        POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
05420
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<460»; }; // NOLINT template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
05421
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
05422
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<460»; }; // NOLINT
05423
              template<> struct ConwayPolynomial<463, 6> { using ZPZ = aerobus::zpz<463>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<462>, ZPZV<51>, ZPZV<10>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<463, 7> { using ZPZ = aerobus::zpz<463>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<460»; }; //
              template<> struct ConwayPolynomial<463, 8> { using ZPZ = aerobus::zpz<463>; using type =
05425
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<414>, ZPZV<396>, ZPZV<3»; }; //
        NOLINT
              template<> struct ConwayPolynomial<463, 9> { using ZPZ = aerobus::zpz<463>; using type =
05426
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<433>, ZPZV<227>, ZPZV<460»;
        }; // NOLINT
05427
              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 =
05428
        POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
05429
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
              template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; }; // NOLINT
05431
             template<> struct ConwayPolynomial<467, 5> { using ZPZ = aerobus::zpz<467>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
05432
              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»; };
           template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<465»; };
          template<> struct ConwayPolynomial<467, 8> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<318>, ZPZV<413>, ZPZV<289>, ZPZV<2»; }; //
      NOLTNT
05435
           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<3>, ZPZV<397>, ZPZV<447>, ZPZV<465»;
       }; // NOLINT
05436
           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 =
05437
      POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; };
                                                     // NOLINT
            template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466»; }; // NOLINT
           template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
05439
      POLYV<ZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<386>, ZPZV<3b; }; // NOLINT template<> struct ConwayPolynomial<479, 5> { using ZPZ = aerobus::zpz<479>; using type =
05440
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<466»; }; // NOLINT
           template<> struct ConwayPolynomial<479, 6> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<243>, ZPZV<243>, ZPZV<243>, ZPZV<334>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<479, 7> { using ZPZ = aerobus::zpz<479>; using type =
05442
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<466»; }; // NOLINT template<> struct ConwayPolynomial<479, 8> { using ZPZ = aerobus::zpz<479>; using type =
05443
       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 =
05444
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<185>, ZPZV<466»; };
       // NOLINT
05445
           template<> struct ConwayPolynomial<487, 1> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<484»; }; // NOLINT
05446
           template<> struct ConwayPolynomial<487, 2> { using ZPZ = aerobus::zpz<487>; using type =
       POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; }; // NOLINT
05447
           template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; }; // NOLINT template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type =
05448
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<484»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 6> { using ZPZ = aerobus::zpz<487>; using type =
05450
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<427>, ZPZV<185>, ZPZV<3»; }; // NOLINT
      template<> struct ConwayPolynomial<487, 7> { using ZPZ = aerobus::zpz<487>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<484%; }; // NOLINT
05451
           template<> struct ConwayPolynomial<487, 8> { using ZPZ = aerobus::zpz<487>; using type =
05452
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<249>, ZPZV<137>, ZPZV<3»; }; //
       NOLINT
05453
           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<1>, ZPZV<27>, ZPZV<271>, ZPZV<447>, ZPZV<484»;
       }; // NOLINT
05454
           template<> struct ConwavPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<489»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
05456
      POLYV<ZPZV<1>, ZPZV<0, ZPZV<2>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360>, ZPZV<2»; }; // NOLINT
05457
           template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
05458
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT
05459
           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 template<> struct ConwayPolynomial<491, 7> { using ZPZ = aerobus::zpz<491>; using type
05460
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 8> { using ZPZ = aerobus::zpz<491>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378>, ZPZV<372>, ZPZV<216>, ZPZV<2»; }; //
       NOLINT
05462
      template<> struct ConwayPolynomial<491, 9> { using ZPZ = aerobus::zpz<491>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<453>, ZPZV<453>, ZPZV<453>,
       }; // NOLINT
05463
            template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<492»; }; // NOLINT
05464
           template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
05465
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; }; // NOLINT
05467
           template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<492»; }; // NOLINT
05468
           template<> struct ConwayPolynomial<499, 6> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<407>, ZPZV<191>, ZPZV<78>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<499,
                                                         7> { using ZPZ = aerobus::zpz<499>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<492x; }; // NOLI template<> struct ConwayPolynomial<499, 8> { using ZPZ = aerobus::zpz<499>; using type
05470
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<288>, ZPZV<309>, ZPZV<200>, ZPZV<7»; }; //
      NOLINT
05471
           template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<491>, ZPZV<222>, ZPZV<492»;
           }; // NOLINT
05472
                 template<> struct ConwayPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
          POLYV<ZPZV<1>, ZPZV<498»; }; // NOLINT
05473
                  template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
          POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; }; // NOLINT
                  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 =
05475
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<325>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<503, 5> { using ZPZ = aerobus::zpz<503>; using type =
05476
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<498»; }; // NOLINT
05477
                  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»; }; // NOLIN
05478
                 template<> struct ConwayPolynomial<503, 7> { using ZPZ = aerobus::zpz<503>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<498»; }; // NoLII template<> struct ConwayPolynomial<503, 8> { using ZPZ = aerobus::zpz<503>; using type =
05479
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, 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<0>, ZPZV<3>, ZPZV<35>, ZPZV<158>, ZPZV<337>, ZPZV<498»;
           }; // NOLINT
05481
                 template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
          POLYV<ZPZV<1>, ZPZV<507»; }; // NOLINT
05482
                  template<> struct ConwayPolynomial<509, 2> { using ZPZ = aerobus::zpz<509>; using type =
           POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
05483
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
05484
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2»; }; // NOLINT

template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
05485
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
                  template<> struct ConwayPolynomial<509, 6> { using ZPZ = aerobus::zpz<509>; using type =
05486
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<350>, ZPZV<232>, ZPZV<41>, ZPZV<2»; }; // NOLINT
05487
                  template<> struct ConwayPolynomial<509, 7> { using ZPZ = aerobus::zpz<509>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<507»; }; // NOLINT
                 template<> struct ConwayPolynomial<509, 8> { using ZPZ = aerobus::zpz<509>; using type =
05488
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<473>, ZPZV<482>, ZPZV<482>, ZPZV<473>, ZPZV<382>, ZPZV<382>, ZPZV<473>, ZPZV<473>, ZPZV<382>, ZPZV<473>, ZPZV
                 template<> struct ConwayPolynomial<509, 9> { using ZPZ = aerobus::zpz<509>; using type =
05489
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50, ZPZV<5
           }; // NOLINT
                  template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
05490
          POLYV<ZPZV<1>, ZPZV<518»; }; // NOLINT
                  template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
          POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; }; // NOLINT
05492
                 template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
          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 =
05493
05494
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<518»; }; // NOLINT
05495
                  template<> struct ConwayPolynomial<521, 6> { using ZPZ = aerobus::zpz<521>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<35>, ZPZV<250>, ZPZV<280>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<521, 7> { using ZPZ = aerobus::zpz<521>; using type :
05496
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<5|8»; }; // NOLINT template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<462>, ZPZV<407>, ZPZV<312>, ZPZV<3*; }; //
05498
                  template<> struct ConwayPolynomial<521, 9> { using ZPZ = aerobus::zpz<521>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<181>, ZPZV<483>, ZPZV<518»;
           }; // NOLINT
05499
                  template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
           POLYV<ZPZV<1>, ZPZV<521»; }; // NOLINT
05500
                 template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
          POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; }; // NOLINT
05501
                  template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<5>, ZPZV<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
05502
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
05503
          05504
                  template<> struct ConwayPolynomial<523, 6> { using ZPZ = aerobus::zpz<523>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<475>, ZPZV<475>, ZPZV<371>, ZPZV<2»; }; // NOLINT
                                                                                          7> { using ZPZ = aerobus::zpz<523>; using type
05505
                  template<> struct ConwayPolynomial<523,
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<521»; }; //
                 template<> struct ConwayPolynomial<523, 8> { using ZPZ = aerobus::zpz<523>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<518>, ZPZV<184>, ZPZV<380>, ZPZV<2»; }; //
           NOLINT
05507
          template<> struct ConwayPolynomial<523, 9> { using ZPZ = aerobus::zpz<523>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<342>, ZPZV<145>, ZPZV<521»;</pre>
          }; // NOLINT
                  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 =
05509
          POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539»; };
                                                            // NOLINT
      template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
05512
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<539»; }; // NOLINT
      template<> struct ConwayPolynomial<541, 6> { using ZPZ = aerobus::zpz<541>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<239>, ZPZV<320>, ZPZV<69>, ZPZV<2»; }; // NOLINT
05513
05514
           template<> struct ConwayPolynomial<541, 7> { using ZPZ = aerobus::zpz<541>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<539»; }; // NOLINT template<> struct ConwayPolynomial<541, 8> { using ZPZ = aerobus::zpz<541>; using type =
05515
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<376>, ZPZV<108>, ZPZV<113>, ZPZV<2»; }; //
      NOLINT
           template<> struct ConwayPolynomial<541, 9> { using ZPZ = aerobus::zpz<541>; using type =
05516
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<340>, ZPZV<318>, ZPZV<539»;
      }; // NOLINT
05517
           template<> struct ConwayPolynomial<547, 1> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<545»; }; // NOLINT
           template<> struct ConwayPolynomial<547, 2> { using ZPZ = aerobus::zpz<547>; using type =
05518
      POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<547, 3> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 4> { using ZPZ = aerobus::zpz<547>; using type =
05520
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<334>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<547, 5> { using ZPZ = aerobus::zpz<547>; using type =
05521
      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<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<2»; }; // NOLINI
           template<> struct ConwayPolynomial<547, 7> { using ZPZ = aerobus::zpz<547>; using type
05523
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<545»; };
           template<> struct ConwayPolynomial<547, 8> { using ZPZ = aerobus::zpz<547>; using type =
05524
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2*; }; //
      NOLINT
           template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type =
05525
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<238>, ZPZV<263>, ZPZV<545»;
      }; // NOLINT
05526
           template<> struct ConwayPolynomial<557, 1> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<555»; }; // NOLINT
           template<> struct ConwayPolynomial<557, 2> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
05528
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<555»; }; // NOLINT template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type =
05529
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<557, 5> { using ZPZ = aerobus::zpz<557>; using type =
05530
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<555»; }; // NOLINT
05531
           template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type =
      05532
           template<> struct ConwayPolynomial<557, 7> { using ZPZ = aerobus::zpz<557>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<555»; }; // NOLINT
           template<> struct ConwayPolynomial<557, 8> { using ZPZ = aerobus::zpz<557>; using type =
05533
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<384>, ZPZV<113>, ZPZV<2»; }; //
05534
           template<> struct ConwayPolynomial<557, 9> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<456>, ZPZV<434>, ZPZV<555»;
      }; // NOLINT
05535
           template<> struct ConwayPolynomial<563, 1> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<561»; }; // NOLINT
           template<> struct ConwayPolynomial<563, 2> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; }; // NOLINT
05537
           template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; ); // NOLINT
template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<20>, ZPZV<399>, ZPZV<2»; }; // NOLINT
05538
           template<> struct ConwayPolynomial<563, 5> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<561»; }; // NOLINT
05540
           template<> struct ConwayPolynomial<563, 6> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<303>, ZPZV<246>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz<563>; using type =
05541
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<561»; }; // NOLINT
           template<> struct ConwayPolynomial<563, 8> { using ZPZ = aerobus::zpz<563>; using type
05542
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<176>, ZPZV<509>, ZPZV<50; }; //
      \label{eq:convergence} $$ \text{template} <> \text{struct ConwayPolynomial} <> 63, 9> { using ZPZ = aerobus::zpz<563>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<15>, ZPZV<19>, ZPZV<561»; }; 
05543
       // NOLINT
           template<> struct ConwayPolynomial<569, 1> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<566»; }; // NOLINT
05545
           template<> struct ConwayPolynomial<569, 2> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
05546
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; ); // NOLINT template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; }; // NOLINT
05548
           template<> struct ConwayPolynomial<569, 5> { using ZPZ = aerobus::zpz<569>; using type =
      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
05549
```

```
template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<566»; }; // NOLINT template<> struct ConwayPolynomial<569, 8> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<527>, ZPZV<173>, ZPZV<241>, ZPZV<241>, ZPZV<3»; }; //
      NOLINT
           template<> struct ConwayPolynomial<569, 9> { using ZPZ = aerobus::zpz<569>; using type =
05552
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<478>, ZPZV<478>, ZPZV<566>, ZPZV<566»;
      }; // NOLINT
05553
           template<> struct ConwayPolynomial<571, 1> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<568»; }; // NOLINT
           template<> struct ConwayPolynomial<571, 2> { using ZPZ = aerobus::zpz<571>; using type =
05554
      POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
05555
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<568»; }; // NOLINT
05556
          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 =
05557
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<568s; }; // NOLINT template<> struct ConwayPolynomial<571, 6> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<221>, ZPZV<295>, ZPZV<33>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<571, 7> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<568»; }; // NOLINT
05560
          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%; }; //
      NOLINT
           template<> struct ConwayPolynomial<571, 9> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<545>, ZPZV<179>, ZPZV<568»;
      }; // NOLINT
05562
           template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<572»; }; // NOLINT
          template<> struct ConwayPolynomial<577, 2> { using ZPZ = aerobus::zpz<577>; using type =
05563
      POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572»; }; // NOLINT
           template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
05565
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<577, 5> { using ZPZ = aerobus::zpz<577>; using type =
05566
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<572»; }; // NOLINT
05567
           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 =
05568
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<572»; }; // NOLINT template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type =
05569
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<5»; }; //
05570
           template<> struct ConwayPolynomial<577, 9> { using ZPZ = aerobus::zpz<577>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>>, ZPZV<576>, ZPZV<449>, ZPZV<572»;
      }; // NOLINT
05571
           template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<585»; }; // NOLINT
           template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2»; }; // NOLINT
05573
          template<> struct ConwayPolynomial<587, 3> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585»; }; // NOLINT template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
05574
      POLYV<ZPZV<1>, ZPZV<3>, ZPZV<16>, ZPZV<444>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<587, 5> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<585»; }; // NOLINT
           template<> struct ConwayPolynomial<587, 6> { using ZPZ = aerobus::zpz<587>; using type =
05576
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<204>, ZPZV<121>, ZPZV<226>, ZPZV<2»; }; // NOLINT
05577
          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<3>, ZPZV<585»; }; // NOLINT
           template<> struct ConwayPolynomial<587, 8> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<492>, ZPZV<44>, ZPZV<91>, ZPZV<2»; };
      template<> struct ConwayPolynomial<587, 9> { using ZPZ = aerobus::zpz<587>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<55>, ZPZV<585»;</pre>
05579
      }; // NOLINT
           template<> struct ConwayPolynomial<593, 1> { using ZPZ = aerobus::zpz<593>; using type =
05580
      POLYV<ZPZV<1>, ZPZV<590»; }; // NOLINT
           template<> struct ConwayPolynomial<593, 2> { using ZPZ = aerobus::zpz<593>; using type =
05581
      POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<593, 3> { using ZPZ = aerobus::zpz<593>; using type =
05582
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<590»; }; // NOLINT template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
05583
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3»; }; // NOLINT
          template<> struct ConwayPolynomial<593, 5> { using ZPZ = aerobus::zpz<593>; using type =
05584
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<590»; }; // NOLINT
05585
           template<> struct ConwayPolynomial<593, 6> { using ZPZ = aerobus::zpz<593>; using type =
      05586
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<590»; }; // NOLINT
           template<> struct ConwayPolynomial<593, 8> { using ZPZ = aerobus::zpz<593>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<291>, ZPZV<495>, ZPZV<495, ZPZV<3»; }; //
      template<> struct ConwayPolynomial<593, 9> { using ZPZ = aerobus::zpz<593>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<223>, ZPZV<523>, ZPZV<590»;</pre>
05588
```

```
}; // NOLINT
05589
            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 =
05590
       POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
05591
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<592»; }; // NOLINT
            template<> struct ConwayPolynomial<599, 4> { using ZPZ = aerobus::zpz<599>; using type =
05592
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<599, 5> { using ZPZ = aerobus::zpz<599>; using type =
05593
      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 =
05594
      POLYV<2PZV<1>, 2PZV<0>, ZPZV<1>, ZPZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592»; }; //
05596
           template<> struct ConwayPolynomial<599, 8> { using ZPZ = aerobus::zpz<599>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<440>, ZPZV<37>, ZPZV<124>, ZPZV<124>, ZPZV<7»; }; //
       NOLINT
           template<> struct ConwayPolynomial<599, 9> { using ZPZ = aerobus::zpz<599>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<114>, ZPZV<98>, ZPZV<592»;
       }; // NOLINT
05598
            template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<594»; }; // NOLINT
           template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
05599
      POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<601, 3> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<594»; }; // NOLINT
           template<> struct ConwayPolynomial<601, 4> { using ZPZ = aerobus::zpz<601>; using type =
05601
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<601, 5> { using ZPZ = aerobus::zpz<601>; using type =
05602
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<594»; }; // NOLINT
05603
            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»; }; // NOLINI
          template<> struct ConwayPolynomial<601, 7> { using ZPZ = aerobus::zpz<601>; using type =
05604
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>4, ZPZV<594*; }; // NOLINT template<> struct ConwayPolynomial<601, 8> { using ZPZ = aerobus::zpz<601>; using type =
05605
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<550>, ZPZV<241>, ZPZV<490>, ZPZV<7»; }; //
           template<> struct ConwayPolynomial<601, 9> { using ZPZ = aerobus::zpz<601>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<487>, ZPZV<590>, ZPZV<594»;
       }; // NOLINT
           template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
05607
      POLYV<ZPZV<1>, ZPZV<604»; }; // NOLINT
05608
           template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; }; // NOLINT
05609
           template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<604»; }; // NOLINT
template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<607, 5> { using ZPZ = aerobus::zpz<607>; using type =
05610
05611
      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 =
05612
       \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 10>, \ \texttt{ZPZV} < 45>, \ \texttt{ZPZV} < 478>, \ \texttt{ZPZV} < 3»; \ \}; \ \ // \ \ \texttt{NOLINT} 
05613
           template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<604»; }; // NOLINT
           template<> struct ConwayPolynomial<607, 8> { using ZPZ = aerobus::zpz<607>; using type =
05614
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<468>, ZPZV<35>, ZPZV<449>, ZPZV<3»; };
      template<> struct ConwayPolynomial<607, 9> { using ZPZ = aerobus::zpz<607>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<444>, ZPZV<444>, ZPZV<4129>, ZPZV<604»;
       }; // NOLINT
05616
            template<> struct ConwayPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<611»; }; // NOLINT
            template<> struct ConwayPolynomial<613, 2> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; }; // NOLINT
05618
           template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<611»; }; // NOLINT template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
05619
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
05620
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<611»; }; // NOLINT
05621
           template<> struct ConwayPolynomial<613, 6> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<609>, ZPZV<595>, ZPZV<601>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type =
05622
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<611»; };
            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»; };
05624
           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<0>, ZPZV<8>, ZPZV<8>, ZPZV<513>, ZPZV<536>, ZPZV<611»;
       }; // NOLINT
            template<> struct ConwayPolynomial<617, 1> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<614»; }; // NOLINT
05626
           template<> struct ConwayPolynomial<617, 2> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<614»; }; // NOLINT
05627
```

```
05628
                           template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type =
               POLYY<ZPZY<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<503>, ZPZV<503>, ZPZV<50; ZPZV<503>, ZPZV<50; ZPZV<503>, ZPZV<50; ZPZV<503>, ZPZV<50; ZPZV<503>, ZPZV<503
05629
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<41, 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
05630
                           template<> struct ConwayPolynomial<617, 7> { using ZPZ = aerobus::zpz<617>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5
                         template<> struct ConwayPolynomial<617, 8> { using ZPZ = aerobus::zpz<617>; using type =
05632
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<519>, ZPZV<501>, ZPZV<155>, ZPZV<3»; }; //
                NOLINT
                          template<> struct ConwayPolynomial<617, 9> { using ZPZ = aerobus::zpz<617>; using type =
05633
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<388>, ZPZV<543>, ZPZV<614»;
                }; // NOLINT
05634
                          template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
               POLYY<ZPZV<1>, ZPZV<6179; }; // NOLINT template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
05635
               POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; }; // NOLINT
                          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 =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492>, ZPZV<2»; }; // NOLINT
                         template<> struct ConwayPolynomial<619, 5> { using ZPZ = aerobus::zpz<619>; using type =
05638
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<617»; }; // NOLINT
05639
                           template<> struct ConwayPolynomial<619, 6> { using ZPZ = aerobus::zpz<619>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<468>, ZPZV<347>, ZPZV<2»; }; // NOLINT
                         template<> struct ConwayPolynomial<619, 7> { using ZPZ = aerobus::zpz<619>; using type =
05640
               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<617»; }; // NOLINT template<> struct ConwayPolynomial<619, 8> { using ZPZ = aerobus::zpz<619>; using type =
05641
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<416>, ZPZV<383>, ZPZV<225>, ZPZV<2»; }; //
                NOLINT
05642
                           template<> struct ConwayPolynomial<619, 9> { using ZPZ = aerobus::zpz<619>; using type
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<310>, ZPZV<617»;
                }; // NOLINT
05643
                          template<> struct ConwayPolynomial<631, 1> { using ZPZ = aerobus::zpz<631>; using type =
               POLYV<ZPZV<1>, ZPZV<628»; }; // NOLINT
                          template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
05644
               POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3»; }; // NOLINT
05645
                           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 =
05646
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<376>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<631, 5> { using ZPZ = aerobus::zpz<631>; using type =
05647
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
                           template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
               POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<516>, ZPZV<541>, ZPZV<106>, ZPZV<3»; }; // NOLINT
05649
                         template<> struct ConwayPolynomial<631, 7> { using ZPZ = aerobus::zpz<631>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<62*, }; // NOLINT
                         template<> struct ConwayPolynomial<631, 8> { using ZPZ = aerobus::zpz<631>; using type =
05650
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<3»; }; //
               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<
05651
                }; // NOLINT
05652
                          template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
               POLYV<ZPZV<1>, ZPZV<638»; }; // NOLINT
                           template<> struct ConwayPolynomial<641, 2> { using ZPZ = aerobus::zpz<641>; using type =
               POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3»; }; // NOLINT
                           template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
05654
              POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<638»; }; // NOLINT
template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<641, 5> { using ZPZ = aerobus::zpz<641>; using type =
05655
05656
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<638»; }; // NOLINT
05657
                         template<> struct ConwayPolynomial<641, 6> { using ZPZ = aerobus::zpz<641>; using type =
                \verb"Polyv<2pzv<1>, & \verb"Zpzv<0>, & \verb"Zpzv<2>, & \verb"Zpzv<105>, & \verb"Zpzv<557>, & \verb"Zpzv<294>, & \verb"Zpzv<3"; & \verb"]; & // & \verb"Nolint" & \verb"Nolint
05658
                          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<3>, ZPZV<638»; }; // NOLINT
                         template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>; using type =
05659
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<3»; }; //
05660
                        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<0>, ZPZV<66>, ZPZV<141>, ZPZV<638»;
                }; // NOLINT
05661
                           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 =
05662
                POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11»; }; // NOLINT
05663
                          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 =
05664
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<600>, ZPZV<11»; };
                                                                                                                                                                     // NOLINT
                           template<> struct ConwayPolynomial<643, 5> { using ZPZ = aerobus::zpz<643>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<632»; }; // NOLINT
05666
                        template<> struct ConwayPolynomial<643, 6> { using ZPZ = aerobus::zpz<643>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<345>, ZPZV<412>, ZPZV<293>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<643, 7> { using ZPZ = aerobus::zpz<643>; using type =
05667
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<632»; };
              template<> struct ConwayPolynomial<643, 8> { using ZPZ = aerobus::zpz<643>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<631>, ZPZV<573>, ZPZV<569>, ZPZV<11»; }; //
        NOLINT
05669
              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<3>, ZPZV<591>, ZPZV<475>, ZPZV<632»;
        }; // NOLINT
05670
              template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<642»; }; // NOLINT
              template<> struct ConwayPolynomial<647, 2> { using ZPZ = aerobus::zpz<647>; using type =
05671
        POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<647, 3> { using ZPZ = aerobus::zpz<647>; using type =
05672
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<6>, ZPZY<642»; }; // NOLINT template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5»; }; // NOLINT
05674
              template<> struct ConwayPolynomial<647, 5> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<642»; }; // NOLINT
        template<> struct ConwayPolynomial<647, 6 { using ZPZ = aerobus::2pz<647>; using type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<308>, ZPZV<385>, ZPZV<642>, ZPZV<5»; }; // NOLINT
05675
              template<> struct ConwayPolynomial<647,
                                                                        7> { using ZPZ = aerobus::zpz<64
                                                                                                                       7>: using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<642»; }; //
05677
              template<> struct ConwayPolynomial<647, 8> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<603>, ZPZV<259>, ZPZV<271>, ZPZV<27»; }; //
        NOLINT
05678
              template<> struct ConwayPolynomial<647, 9> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<561>, ZPZV<123>, ZPZV<642»;
        }; // NOLINT
05679
              template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<651»; }; // NOLINT
              template<> struct ConwayPolynomial<653, 2> { using ZPZ = aerobus::zpz<653>; using type =
05680
        POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2»; }; // NOLINT
05681
              template<> struct ConwayPolynomial<653, 3> { using ZPZ = aerobus::zpz<653>; using type =
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZV<651»; }; // NOLINT template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type =
05682
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
05683
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651»; }; // NOLINT
              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<24>; }; // NOLINT
              template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<651»; };
05686
              template<> struct ConwayPolynomial<653, 8> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2»; };
              template<> struct ConwayPolynomial<653, 9> { using ZPZ = aerobus::zpz<653>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65)»;
         }; // NOLINT
05688
              template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<657»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
05689
        POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<657»; }; // NOLINT template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
05691
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
05692
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<657»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 6> { using ZPZ = aerobus::zpz<659>; using type =
05693
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<65, ZPZV<371>, ZPZV<205>, ZPZV<23>, ZPZV<22*; }; // NOLINT template<> struct ConwayPolynomial<659, 7> { using ZPZ = aerobus::zpz<659>; using type =
05694
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<657»; }; // NOLINT template<> struct ConwayPolynomial<659, 8> { using ZPZ = aerobus::zpz<659>; using type =
05695
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<358>, ZPZV<246>, ZPZV<90>, ZPZV<2»; };
05696
             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<46>, ZPZV<657»;
        }; // NOLINT
05697
              \texttt{template<> struct ConwayPolynomial<661, 1> \{ using \ ZPZ = aerobus:: zpz<661>; using \ type = aerobus:: zpz<661>; usi
        POLYV<ZPZV<1>, ZPZV<659»; }; // NOLINT
05698
              template<> struct ConwayPolynomial<661, 2> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2»; }; // NOLINT
05699
              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 =
05700
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<659»; }; // NOLINT
05702
              template<> struct ConwayPolynomial<661, 6> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<456>, ZPZV<382>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<661, 7> { using ZPZ = aerobus::zpz<661>; using type =
05703
        POLYV<ZPZV<1>, 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 =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<612>, ZPZV<285>, ZPZV<72>, ZPZV<72>; };
        NOLINT
05705
              template<> struct ConwayPolynomial<661, 9> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<389>, ZPZV<220>, ZPZV<659»;
        }; // NOLINT
```

```
template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<668»; }; // NOLINT
05707
           template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5»; }; // NOLINT
05708
            template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5»; }; // NOLINT
05710
           template<> struct ConwayPolynomial<673, 5> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 6> { using ZPZ = aerobus::zpz<673>; using type =
05711
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<524>, ZPZV<35>, ZPZV<35>; // NOLINT template<> struct ConwayPolynomial<673, 7> { using ZPZ = aerobus::zpz<673>; using type
05712
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<668»; }; //
05713
           template<> struct ConwayPolynomial<673, 8> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<669>, ZPZV<587>, ZPZV<302>, ZPZV<5»; }; //
       template<> struct ConwayPolynomial<673, 9> { using ZPZ = aerobus::zpz<673>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<347>, ZPZV<347>, ZPZV<553>, ZPZV<668»;
05714
            template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
05715
       POLYV<ZPZV<1>, ZPZV<675»; }; // NOLINT
           template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
0.5716
       POLYV<ZPZV<1>, ZPZV<672>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<677, 3> { using ZPZ = aerobus::zpz<677>; using type =
05717
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<675»; }; // NOLINT template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
05718
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<677, 5> { using ZPZ = aerobus::zpz<677>; using type =
05719
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<5>, ZPZV<67, $PZV<67, $PZV<67, $PZV<67, $PZV<67, $PZV<67, $PZV<67, $PZV<67, $PZV<675»; }; // NOLINT template<> struct ConwayPolynomial<677, 6> { using ZPZ = aerobus::zpz<677>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446>, ZPZV<632>, ZPZV<50>, ZPZV<2»; }; // NOLINT
05720
            template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type =
05721
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<675»; };
           template<> struct ConwayPolynomial<677, 8> { using ZPZ = aerobus::zpz<677>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<61>, ZPZV<619>, ZPZV<619>, ZPZV<152>, ZPZV<2»; }; //
            template<> struct ConwayPolynomial<677, 9> { using ZPZ = aerobus::zpz<677>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<504>, ZPZV<404>, ZPZV<675»;
       }; // NOLINT
05724
            template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
       POLYV<ZPZV<1>, ZPZV<678»; }; // NOLINT
            template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
05725
       POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; }; // NOLINT
            template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<678»; }; // NOLINT template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type =
05727
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<683, 5> { using ZPZ = aerobus::zpz<683>; using type =
05728
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<678»; }; // NOLINT
            template<> struct ConwayPolynomial<683, 6> { using ZPZ = aerobus::zpz<683>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<644>, ZPZV<109>, ZPZV<434>, ZPZV<5»; }; // NOLINT
05730
           template<> struct ConwayPolynomial<683, 7> { using ZPZ = aerobus::zpz<683>; using type =
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<678»; }; // NOLI template<> struct ConwayPolynomial<683, 8> { using ZPZ = aerobus::zpz<683>; using type =
                                                                                                                // NOLTNT
05731
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<5»; };
05732
           template<> struct ConwayPolynomial<683, 9> { using ZPZ = aerobus::zpz<683>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<85>, ZPZV<444>, ZPZV<678»;
       }; // NOLINT
05733
            template<> struct ConwayPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
       POLYV<ZPZV<1>, ZPZV<688»; }; // NOLINT
05734
            template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
       POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; }; // NOLINT
05735
           template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<688»; }; // NOLINT
template<> struct ConwayPolynomial<691, 4> { using ZPZ = aerobus::zpz<691>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<691, 5> { using ZPZ = aerobus::zpz<691>; using type =
05736
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; }; // NOLINT
            template<> struct ConwayPolynomial<691, 6> { using ZPZ = aerobus::zpz<691>; using type =
05738
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<408>, ZPZV<262>, ZPZV<3»; }; // NOLINT
05739
            template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<48, ZPZV<688»; }; // NOLINT template<> struct ConwayPolynomial<691, 8> { using ZPZ = aerobus::zpz<691>; using type =
05740
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<356>, ZPZV<425>, ZPZV<425>, ZPZV<321>, ZPZV<3»; }; //
05741
           template<> struct ConwayPolynomial<691, 9> { using ZPZ = aerobus::zpz<691>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<556>, ZPZV<443>, ZPZV<688»;
       }; // NOLINT
05742
            template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
       POLYV<ZPZV<1>, ZPZV<699»; }; // NOLINT
            template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
05744
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; }; // NOLINT template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2»; };
           template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699»; }; // NOLINT
05747
           template<> struct ConwayPolynomial<701, 6> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<571>, ZPZV<327>, ZPZV<285>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<701, 7> { using ZPZ = aerobus::zpz<701>; using type
05748
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<699»; }; // NOLINT
05749
           template<> struct ConwayPolynomial<701, 8> { using ZPZ = aerobus::zpz<701>;
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<619>, ZPZV<206>, ZPZV<593>, ZPZV<2»; }; //
      template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<45>, ZPZV<45>, ZPZV<45>, ZPZV<373>, ZPZV<699»;
05750
      }; // NOLINT
  template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<707»; }; // NOLINT
05752
           template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
05753
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707»; }; // NOLINT
           template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; }; // NOLINT
05755
           template<> struct ConwayPolynomial<709, 5> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<707»; }; // NOLINT template<> struct ConwayPolynomial<709, 6> { using ZPZ = aerobus::zpz<709>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<669>, ZPZV<514>, ZPZV<295>, ZPZV<2»; }; // NOLINT
05756
           template<> struct ConwayPolynomial<709, 7> { using ZPZ = aerobus::zpz<709>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<707»; };
05758
           template<> struct ConwayPolynomial<709, 8> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<689>, ZPZV<233>, ZPZV<79>, ZPZV<2»; }; //
      NOLINT
           template<> struct ConwayPolynomial<709, 9> { using ZPZ = aerobus::zpz<709>; using type =
05759
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<257>, ZPZV<277), ZPZV<707»;
       }; // NOLINT
           template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
05760
      POLYV<ZPZV<1>, ZPZV<708»; }; // NOLINT
           template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
05761
      POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
           template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
05763
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<602>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<719, 5> { using ZPZ = aerobus::zpz<719>; using type =
05764
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<708»; }; // NOLINT
05765
           template<> struct ConwayPolynomial<719, 6> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<533>, ZPZV<591>, ZPZV<182>, ZPZV<11»; }; // NOLINT
05766
           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<0>, ZPZV<1>, ZPZV<11>, ZPZV<708»; }; // NOLII template<> struct ConwayPolynomial<719, 8> { using ZPZ = aerobus::zpz<719>; using type =
05767
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<714>, ZPZV<362>, ZPZV<244>, ZPZV<11»; }; //
      NOLINT
05768
           template<> struct ConwayPolynomial<719, 9> { using ZPZ = aerobus::zpz<719>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<288>, ZPZV<560>, ZPZV<708»;
       }; // NOLINT
05769
           template<> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<722»; }; // NOLINT
           template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
05770
       POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<722»; }; // NOLINT
template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
05772
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<727, 5> { using ZPZ = aerobus::zpz<727>; using type =
05773
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<722»; }; // NOLINT
            template<> struct ConwayPolynomial<727, 6> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<86>, ZPZV<397>, ZPZV<672>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<727, 7> { using ZPZ = aerobus::zpz<727>; using type =
05775
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<722»; }; // NOLINT template<> struct ConwayPolynomial<727, 8> { using ZPZ = aerobus::zpz<727>; using type =
05776
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<671>, ZPZV<368>, ZPZV<36»; }; //
      NOLINT
           template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type =
05777
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<573>, ZPZV<502>, ZPZV<722»;
       }; // NOLINT
05778
           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 =
      POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; }; // NOLINT
05780
           template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727»; }; // NOLINT template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
0.5781
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<6s; }; // NOLINT template<> struct ConwayPolynomial<733, 5> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<727»; }; // NOLINT
05783
           template<> struct ConwayPolynomial<733, 6> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<549>, ZPZV<151>, ZPZV<6»; }; // NOLINT 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<3>, ZPZV<727»; }; // NOLINT
05784
```

```
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»; };</pre>
05786
         // NOLINT
               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 =
05788
        POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
05789
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; }; // NOLINT template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
05790
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<3»; }; // NOLINT
05791
              template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<736»; }; // NOLINT template<> struct ConwayPolynomial<739, 6> { using ZPZ = aerobus::zpz<739>; using type =
05792
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<422>, ZPZV<47>, ZPZV<625, ZPZV<3»; ; // NOLINT 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<0>, ZPZV<44>, ZPZV<736»; };
              template<> struct ConwayPolynomial<739, 8> { using ZPZ = aerobus::zpz<739>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<401>, ZPZV<169>, ZPZV<25>, ZPZV<3»; };
         NOLINT
        template<> struct ConwayPolynomial<739, 9> { using ZPZ = aerobus::zpz<739>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<616>, ZPZV<81>, ZPZV<81>, ZPZV<736»;</pre>
05795
         }; // NOLINT
              template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
05796
        POLYV<ZPZV<1>, ZPZV<738»; }; // NOLINT
05797
               template<> struct ConwayPolynomial<743, 2> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<743, 3> { using ZPZ = aerobus::zpz<743>; using type =
05798
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<738»; }; // NOLINT
               template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
05799
        0.5800
               template<> struct ConwayPolynomial<743, 5> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT
        template<> struct ConwayPolynomial</a>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</pre>
05801
05802
               template<> struct ConwayPolynomial<743, 7> { using ZPZ = aerobus::zpz<743>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 >, ZP
05803
              template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<279>, ZPZV<588>, ZPZV<5»; }; //
         NOLINT
05804
              template<> struct ConwayPolynomial<743, 9> { using ZPZ = aerobus::zpz<743>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<327>, ZPZV<676>, ZPZV<738»;
         }; // NOLINT
05805
              template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
        POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
               template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
05806
        POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
05807
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; }; // NOLINT
05808
               template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<525>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<751, 5> { using ZPZ = aerobus::zpz<751>; using type =
05809
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT template<> struct ConwayPolynomial<751, 6> { using ZPZ = aerobus::zpz<751>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<298>, ZPZV<633>, ZPZV<539>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<751, 7> { using ZPZ = aerobus::zpz<751>; using type =
05811
        POLYV<ZPZV<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 =
05812
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<741>, ZPZV<243>, ZPZV<672>, ZPZV<3»; }; //
         NOLINT
               template<> struct ConwayPolynomial<751, 9> { using ZPZ = aerobus::zpz<751>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<703>, ZPZV<489>, ZPZV<748»;
         }; // NOLINT
05814
               template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
        POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
               template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
05815
        POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
05816
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
05817
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<757, 5> { using ZPZ = aerobus::zpz<757>; using type =
05818
        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 =
05819
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<753>, ZPZV<739>, ZPZV<745>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<757, 7> { using ZPZ = aerobus::zpz<757>; using type =
05820
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<755»; }; // NOLINT
              template<> struct ConwayPolynomial<757, 8> { using ZPZ = aerobus::zpz<757>; using type
05821
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<110>, ZPZV<509>, ZPZV<2»; }; //
05822
              template<> struct ConwayPolynomial<757, 9> { using ZPZ = aerobus::zpz<757>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<88>, ZPZV<688>, ZPZV<702>, ZPZV<755»;
         }; // NOLINT
05823
              template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<755»; };
                 template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
          POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
05825
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<6»; }; // NOLINT
05826
                 template<> struct ConwayPolynomial<761, 5> { using ZPZ = aerobus::zpz<761>; using type =
05827
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT
05828
                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 = DOLYMARPRIACO | RDZW20 |
05829
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<755»; };
                 template<> struct ConwayPolynomial<761, 8> { using ZPZ = aerobus::zpz<761>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<603>, ZPZV<144>, ZPZV<540>, ZPZV<540>, }; //
                template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type =
05831
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<317>, ZPZV<571>, ZPZV<755»;
          }; // NOLINT
                 template<> struct ConwayPolynomial<769, 1> { using ZPZ = aerobus::zpz<769>; using type =
          POLYV<ZPZV<1>, ZPZV<758»; }; // NOLINT
                 template<> struct ConwayPolynomial<769, 2> { using ZPZ = aerobus::zpz<769>; using type =
05833
          POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; }; // NOLINT
                 template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
05834
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; }; // NOLINT
                 template<> struct ConwayPolynomial<769, 4> { using ZPZ = aerobus::zpz<769>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; }; // NOLINT
                 template<> struct ConwayPolynomial<769, 5> { using ZPZ = aerobus::zpz<769>; using type =
05836
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<758»; }; // NOLINT
05837
                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<326>, ZPZV<650>, ZPZV<11»; }; // NOLINT
05838
                 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»; }; //
05839
                template<> struct ConwayPolynomial<769, 8> { using ZPZ = aerobus::zpz<769>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<560>, ZPZV<574>, ZPZV<632>, ZPZV<11»; }; //
          NOLINT
          template<> struct ConwayPolynomial<769, 9> { using ZPZ = aerobus::zpz<769>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<623>, ZPZV<751>, ZPZV<7
05840
05841
                 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 =
05842
          POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»: }: // NOLINT
05843
                 template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<771»; }; // NOLINT template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
05844
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2»; }; // NOLINT
                template<> struct ConwayPolynomial<773, 5> { using ZPZ = aerobus::zpz<773>; using type =
05845
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<771»; }; // NOLINT
                 template<> struct ConwayPolynomial<773, 6> { using ZPZ = aerobus::zpz<773>; using type =
05846
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<91>, ZPZV<3>, ZPZV<581>, ZPZV<2»; };
                 template<> struct ConwayPolynomial<773, 7> { using ZPZ = aerobus::zpz<773>; using type =
05847
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<771»; }; // NOLINT
05848
                template<> struct ConwayPolynomial<773, 8> { using ZPZ = aerobus::zpz<773>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<484>, ZPZV<94>, ZPZV<693>, ZPZV<2»; };
          NOLINT
                 template<> struct ConwayPolynomial<773, 9> { using ZPZ = aerobus::zpz<773>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<216>, ZPZV<574>, ZPZV<771»;
          }; // NOLINT
05850
                 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 =
05851
          POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785»; }; // NOLINT
                template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type =
05853
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
05854
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<785»; }; // NOLINT
                 template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
05855
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<98>, ZPZV<512>, ZPZV<606>, ZPZV<2»; }; // NOLINT
05856
                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<3>, ZPZV<785»; }; // NOLINT template<> struct ConwayPolynomial<787, 8> { using ZPZ = aerobus::zpz<787>; using type =
05857
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<612>, ZPZV<26>, ZPZV<715>, ZPZV<2»; }; //
05858
                template<> struct ConwayPolynomial<787, 9> { using ZPZ = aerobus::zpz<787>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<480>, ZPZV<573>, ZPZV<785»;
          }; // NOLINT
05859
                 template<> struct ConwayPolynomial<797, 1> { using ZPZ = aerobus::zpz<797>; using type =
          POLYV<ZPZV<1>, ZPZV<795»; }; // NOLINT
                 template<> struct ConwayPolynomial<797, 2> { using ZPZ = aerobus::zpz<797>; using type =
          POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»; }; // NOLINT
05861
                template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<717>, ZPZV<2»; }; // NOLINT
05862
```

```
template<> struct ConwayPolynomial<797, 5> { using ZPZ = aerobus::zpz<797>; using type =
05863
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<795»; }; // NOLINT
05864
              template<> struct ConwayPolynomial<797, 6> { using ZPZ = aerobus::zpz<797>; using type =
        POLYY<ZPZV<1>, ZPZV<3>, ZPZV<1>, ZPZV<657>, ZPZV<657>, ZPZV<71>, ZPZV<22; }; // NOLINT template<> struct ConwayPolynomial<797, 7> { using ZPZ = aerobus::zpz<797>; using type :
05865
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<795»; };
                                                                                                                                        // NOLINT
               template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<38), //
        template<> struct ConwayPolynomial<797, 9> { using ZPZ = aerobus::zpz<797>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<240>, ZPZV<240>, ZPZV<599>, ZPZV<795»;</pre>
05867
         }; // NOLINT
05868
               template<> struct ConwayPolynomial<809, 1> { using ZPZ = aerobus::zpz<809>; using type =
         POLYV<ZPZV<1>, ZPZV<806»; }; // NOLINT
05869
              template<> struct ConwayPolynomial<809, 2> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
05870
               template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<806»; }; // NOLINT
               template<> struct ConwayPolynomial809, 4> { using ZPZ = aerobus::zpz<809>; using type =
05871
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<809, 5> { using ZPZ = aerobus::zpz<809>; using type =
05872
        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 =
05873
        POLYVZPZVX1>, ZPZV<0>, ZPZV<1>, ZPZV<562>, ZPZV<75>, ZPZV<43>, ZPZV<3»; ); // NOLINT template<> struct ConwayPolynomial<809, 7> { using ZPZ = aerobus::zpz<809>; using type
05874
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; }; // NOLINT
05875
              template<> struct ConwayPolynomial<809, 8> { using ZPZ = aerobus::zpz<809>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<593>, ZPZV<745>, ZPZV<673>, ZPZV<673>, ZPZV<3»; }; //
         NOLINT
05876
              template<> struct ConwayPolynomial<809, 9> { using ZPZ = aerobus::zpz<809>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<727>, ZPZV<806»;
         }; // NOLINT
              template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
05877
        POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
               template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
05878
        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
05879
05880
               template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type =
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<453>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<811, 5> { using ZPZ = aerobus::zpz<811>; using type =
05881
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<808*; }; // NOLINT template<> struct ConwayPolynomial<811, 6> { using ZPZ = aerobus::zpz<811>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<780>, ZPZV<755>, ZPZV<307>, ZPZV<30*; }; // NOLINT
05882
               template<> struct ConwayPolynomial<811, 7> { using ZPZ = aerobus::zpz<811>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<808»; };
05884
              template<> struct ConwayPolynomial<811, 8> { using ZPZ = aerobus::zpz<811>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<3»; }; //
         NOLINT
               template<> struct ConwayPolynomial<811, 9> { using ZPZ = aerobus::zpz<811>; using type =
05885
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<382>, ZPZV<200>, ZPZV<808»;
         }; // NOLINT
05886
               template<> struct ConwayPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<819»; }; // NOLINT template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
05887
        POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<array>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 =
05889
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<662>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
05890
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<819»; }; // NOLINT
05891
               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
05892
              template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<819»; }; // NOLINT template<> struct ConwayPolynomial<821, 8> { using ZPZ = aerobus::zpz<821>; using type =
05893
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<556>, ZPZV<589>, ZPZV<2»; }; //
         NOLINT
               template<> struct ConwayPolynomial<821, 9> { using ZPZ = aerobus::zpz<821>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<650>, ZPZV<557>, ZPZV<819»;
         }; // NOLINT
05895
               template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
               template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
05896
        POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
05897
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
05898
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
05899
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
               template<> struct ConwayPolynomial<823, 6> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<822>, ZPZV<616>, ZPZV<744>, ZPZV<3»; }; // NOLINT
05901
              template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type =
        POLYVCZPZVC1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2
05902
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<451>, ZPZV<437>, ZPZV<31>, ZPZV<3»; }; //
05903
                 template<> struct ConwayPolynomial<823, 9> { using ZPZ = aerobus::zpz<823>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<740>, ZPZV<609>, ZPZV<820»;
           }; // NOLINT
                  template<> struct ConwayPolynomial<827, 1> { using ZPZ = aerobus::zpz<827>; using type =
05904
          POLYV<ZPZV<1>, ZPZV<825»; }; // NOLINT
                  template<> struct ConwayPolynomial<827, 2> { using ZPZ = aerobus::zpz<827>; using type =
05905
           POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT
05906
                 template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825,; }; // NOLINT template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type =
05907
          POLYVCZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605, ZPZV<605, ZPZV<605, ZPZV<27; ); // NOLINT template<> struct ConwayPolynomial<827, 5> { using ZPZ = aerobus::zpz<827>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<825»; }; // NOLINT
05909
                 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<691>, ZPZV<691>; // NOLINT template<> struct ConwayPolynomial<827, 7> { using ZPZ = aerobus::zpz<827>; using type
05910
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<825»; }; // NOLINT
05911
                  template<> struct ConwayPolynomial<827, 8> { using ZPZ = aerobus::zpz<827>; using type =
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<812>, ZPZV<79>, ZPZV<32>, ZPZV<2»; };
           NOLINT
05912
                 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
                  template<> struct ConwayPolynomial<829, 1> { using ZPZ = aerobus::zpz<829>; using type =
          POLYV<ZPZV<1>, ZPZV<827»; }; // NOLINT
05914
                  template<> struct ConwayPolynomial<829, 2> { using ZPZ = aerobus::zpz<829>; using type =
          POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
05915
          POLYVCZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<62, ZPZV<827»; }; // NOLINT template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
05916
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<604>, ZPZV<2»; }; // NOLINT
05917
                 template<> struct ConwayPolynomial<829, 5> { using ZPZ = aerobus::zpz<829>; using type =
           \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<7>, \verb"ZPZV<827"; \verb"}; $ // \verb"NOLINT" | NOLINT" 
                  template<> struct ConwayPolynomial<829, 6> { using ZPZ = aerobus::zpz<829>; using type =
05918
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<345, ZPZV<817>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<827»; };
                 template<> struct ConwayPolynomial<829, 8> { using ZPZ = aerobus::zpz<829>; using type =
05920
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<468>, ZPZV<241>, ZPZV<138>, ZPZV<2»; }; //
           NOLINT
                 template<> struct ConwayPolynomial<829, 9> { using ZPZ = aerobus::zpz<829>; using type =
05921
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<621>, ZPZV<6252>, ZPZV<827»;
05922
                  template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
          POLYV<ZPZV<1>, ZPZV<828»; }; // NOLINT
                 template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
05923
          POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; }; // NOLINT
                  template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
05924
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; }; // NOLINT
                  template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
           \verb"POLYV<ZPZV<1>, \ \verb"ZPZV<0>, \ \verb"ZPZV<2>, \ \verb"ZPZV<609>, \ \verb"ZPZV<11"; \ \verb"}; \ \ // \ \verb"NOLINT" 
05926
                 template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<828; }; // NOLINT
                  template<> struct ConwayPolynomial<839, 6> { using ZPZ = aerobus::zpz<839>; using type =
05927
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<370>, ZPZV<537>, ZPZV<23>, ZPZV<11»; }; // NOLINT
                 template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type
05928
          POLYV<ZPZV<1>, 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 =
05929
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<553>, ZPZV<779>, ZPZV<329>, ZPZV<11»; }; //
           NOLINT
05930
                  template<> struct ConwayPolynomial<839, 9> { using ZPZ = aerobus::zpz<839>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<349>, ZPZV<206>, ZPZV<828*;
           }; // NOLINT
05931
                 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 =
05932
          POLYV<ZPZV<1>, ZPZV<852>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<853, 3> { using ZPZ = aerobus::zpz<853>; using type =
05933
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
05934
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<623>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
05935
          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 =
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<276>, ZPZV<194>, ZPZV<512>, ZPZV<2»; }; // NOLINT
05937
                 template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
05938
          POLYV<ZPZV<1>, ZPZV<0>, 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>; ZPZV<877>, ZPZV<821>, ZPZV<851»;
           }; // NOLINT
                  template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
05940
           POLYV<ZPZV<1>, ZPZV<854»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
           POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; }; // NOLINT
05942
                  template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; }; // NOLINT
template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<528>, ZPZV<3»; }; // NOLINT
05943
                   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 =
05945
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<82>, ZPZV<824>, ZPZV<65>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::zpz<857>; using type =
05946
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<854»; }; // NOLINT
                  template<> struct ConwayPolynomial<857, 8> { using ZPZ = aerobus::zpz<857>; using type =
05947
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<611>, ZPZV<552>, ZPZV<494>, ZPZV<3»; }; //
           NOLINT
05948
                  template<> struct ConwayPolynomial<857, 9> { using ZPZ = aerobus::zpz<857>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<719>, ZPZV<854»;
           }; // NOLINT
                   template<> struct ConwayPolynomial<859, 1> { using ZPZ = aerobus::zpz<859>; using type =
           POLYV<ZPZV<1>, ZPZV<857»; }; // NOLINT
                   template<> struct ConwayPolynomial<859, 2> { using ZPZ = aerobus::zpz<859>; using type =
           POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
05951
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type =
05952
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<859, 5> { using ZPZ = aerobus::zpz<859>; using type =
05953
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<857»; }; // NOLINT
05954
                   template<> struct ConwayPolynomial<859, 6> { using ZPZ = aerobus::zpz<859>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<419>, ZPZV<646>, ZPZV<566>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpz<859>; using type
05955
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<857»; };
                  template<> struct ConwayPolynomial<859, 8> { using ZPZ = aerobus::zpz<859>; using type =
05956
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<522>, ZPZV<446>, ZPZV<672>, ZPZV<62»; }; //
           NOLINT
05957
                  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
05958
                   template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
           POLYV<ZPZV<1>, ZPZV<858»; }; // NOLINT
05959
                  template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
           POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
05960
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<588%; }; // NOLINT template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5»; }; // NOLINT
          template<> struct ConwayPolynomial<863, 5> { using ZPZ = aerobus::zpz<863>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<858»; }; // NOLINT</pre>
05962
           template<> struct ConwayPolynomial<863, 6> { using ZPZ = aerobus::zpz<863>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<30>, ZPZV<62>, ZPZV<300>, ZPZV<5»; }; // NOLINT
05963
05964
                  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<858»; }; // NOLIN template<> struct ConwayPolynomial<863, 8> { using ZPZ = aerobus::zpz<863>; using type =
05965
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<765>, ZPZV<576>, ZPZV<849>, ZPZV
           NOLINT
           template<> struct ConwayPolynomial<863, 9> { using ZPZ = aerobus::zpz<863>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<381>, ZPZV<1>, ZPZV<858»; };</pre>
05966
                   template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
05967
           POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
                  template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
05968
           POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; }; // NOLINT
05969
                   template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<5>, ZPZY<5>, ZPZY<75, ZPZY<5>, ZPZY<5
05970
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<877, 5> { using ZPZ = aerobus::zpz<877>; using type =
05971
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>; }; // NOLINT template<> struct ConwayPolynomial<877, 6> { using ZPZ = aerobus::zpz<877>; using type =
05972
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<400>, ZPZV<855>, ZPZV<855>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<877, 7> { using ZPZ = aerobus::zpz<877>; using type =
05973
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<875»; }; // NOLINT
05974
                  template<> struct ConwayPolynomial<877, 8> { using ZPZ = aerobus::zpz<877>; using type =
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<767>, ZPZV<319>, ZPZV<347>, ZPZV<2»: }; //
           NOLINT
                  template<> struct ConwayPolynomial<877, 9> { using ZPZ = aerobus::zpz<877>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<770>, ZPZV<78>, ZPZV<875»;
           }; // NOLINT
05976
                  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 =
05977
          POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<878»; };
                                                                                                     // NOLINT
                 template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
05979
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<447>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<881, 5> { using ZPZ = aerobus::zpz<881>; using type =
05980
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>; };
            template<> struct ConwayPolynomial<881, 6> { using ZPZ = aerobus::zpz<881>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<419>, ZPZV<231>, ZPZV<3»; }; // NOLINT
05982
           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<6>, ZPZV<6>, ZPZV<6>, ZPZV<878»; }; // NOLINT template<> struct ConwayPolynomial<881, 8> { using ZPZ = aerobus::zpz<881>; using type =
05983
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<635>, ZPZV<490>, ZPZV<561>, ZPZV<561>, ZPZV<3»; }; //
05984
           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»;
       }; // NOLINT
            template<> struct ConwayPolynomial<883, 1> { using ZPZ = aerobus::zpz<883>; using type =
05985
       POLYV<ZPZV<1>, ZPZV<881»; }; // NOLINT
            template<> struct ConwayPolynomial<883, 2> { using ZPZ = aerobus::zpz<883>; using type =
       POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; }; // NOLINT
05987
           template<> struct ConwayPolynomial<883, 3> { using ZPZ = aerobus::zpz<883>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<881»; }; // NOLINT template<> struct ConwayPolynomial<883, 4> { using ZPZ = aerobus::zpz<883>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2»; }; // NOLINT
05988
           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
05990
           template<> struct ConwayPolynomial<883, 6> { using ZPZ = aerobus::zpz<883>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<87>>, ZPZV<865>, ZPZV<871>, ZPZV<2x; }; // NOLINT template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type
05991
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<8>, ZPZV<881»; };
           template<> struct ConwayPolynomial<883, 8> { using ZPZ = aerobus::zpz<883>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<740>, ZPZV<762>, ZPZV<768>, ZPZV<768>, ZPZV<2»; }; //
       NOLINT
       template<> struct ConwayPolynomial<883, 9> { using ZPZ = aerobus::zpz<883>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<360>, ZPZV<360>, ZPZV<881»;</pre>
05993
       }; // NOLINT
05994
            template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
       POLYV<ZPZV<1>, ZPZV<882»; }; // NOLINT
05995
           template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
       POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
05996
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<882»; }; // NOLINT template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<887, 5> { using ZPZ = aerobus::zpz<887>; using type =
05998
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<882»; }; // NOLINT
05999
           template<> struct ConwayPolynomial<887, 6> { using ZPZ = aerobus::zpz<887>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<775>, ZPZV<341>, ZPZV<28>, ZPZV<5»; }; // NOLINT
06000
           template<> struct ConwayPolynomial<887,
                                                           7> { using ZPZ = aerobus::zpz<887>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<882»; }; // NOLINT
06001
           template<> struct ConwayPolynomial<887, 8> { using ZPZ = aerobus::zpz<887>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<781>, ZPZV<381>, ZPZV<706>, ZPZV<70s, }; //
       NOLINT
06002
           template<> struct ConwayPolynomial<887, 9> { using ZPZ = aerobus::zpz<887>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<727>, ZPZV<345>, ZPZV<882»;
       }; // NOLINT
    template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
06003
       POLYY<ZPZY<1>, ZPZY<905»; }; // NOLINT template<> struct ConwayPolynomial<907, 2> { using ZPZ = aerobus::zpz<907>; using type =
06004
       POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<907, 3> { using ZPZ = aerobus::zpz<907>; using type =
06005
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<905»; }; // NOLINT
           template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
06006
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
06007
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905; }; // NOLINT template<> struct ConwayPolynomial<907, 6> { using ZPZ = aerobus::zpz<907>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<626>, ZPZV<266>, ZPZV<266>, ZPZV<28; }; // NOLINT
06008
           template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<905»; };
06010
           template<> struct ConwayPolynomial<907, 8> { using ZPZ = aerobus::zpz<907>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<584>, ZPZV<518>, ZPZV<811>, ZPZV<81; }; //
       NOLINT
06011
           template<> struct ConwayPolynomial<907, 9> { using ZPZ = aerobus::zpz<907>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<783>, ZPZV<575, ZPZV<905»;
       }; // NOLINT
06012
           template<> struct ConwayPolynomial<911, 1> { using ZPZ = aerobus::zpz<911>; using type =
       POLYV<ZPZV<1>, ZPZV<894»; }; // NOLINT
           template<> struct ConwayPolynomial<911, 2> { using ZPZ = aerobus::zpz<911>; using type =
06013
       POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; };
                                                      // NOLINT
            template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; }; // NOLINT
           template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
06015
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
06016
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<894»; }; // NOLINT
            template<> struct ConwayPolynomial<911, 6> { using ZPZ = aerobus::zpz<911>; using type =
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<172>, ZPZV<683>, ZPZV<19>, ZPZV<17*; }; // NOLINT template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
06018
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<49, ZPZV<894»; }; // NOLINT template<> struct ConwayPolynomial<911, 8> { using ZPZ = aerobus::zpz<911>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<590>, ZPZV<168>, ZPZV<17»; }; //
06019
```

```
NOLINT
         template<> struct ConwayPolynomial<911, 9> { using ZPZ = aerobus::zpz<911>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<679>, ZPZV<616>, ZPZV<894»;
06020
         }; // NOLINT
06021
                template<> struct ConwayPolynomial<919, 1> { using ZPZ = aerobus::zpz<919>; using type =
         POLYV<ZPZV<1>, ZPZV<912»; }; // NOLINT
                template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
         POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; }; // NOLINT
               template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
06023
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912»; }; // NOLINT template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
06024
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<919, 5> { using ZPZ = aerobus::zpz<919>; using type =
06025
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<912»; }; // NOLINT
06026
              template<> struct ConwayPolynomial<919, 6> { using ZPZ = aerobus::zpz<919>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<312>, ZPZV<817>, ZPZV<113>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<919, 7> { using ZPZ = aerobus::zpz<919>; using type =
06027
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<202>, ZPZV<504>, ZPZV<7*; }; //
06029
               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<7>, ZPZV<410>, ZPZV<623>, ZPZV<912»;
         }; // NOLINT
06030
                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 =
06031
         POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3»; }; // NOLINT
06032
               template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<926»; }; // NOLINT template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3»; }; // NOLINT
06033
               template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
06034
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<926»; }; // NOLINT
06035
               template<> struct ConwayPolynomial<929, 6> { using ZPZ = aerobus::zpz<929>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<805>, ZPZV<86>, ZPZV<86>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type =
06036
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<926»; }; // NOLINT
               template<> struct ConwayPolynomial<929, 8> { using ZPZ = aerobus::zpz<929>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699>, ZPZV<292>, ZPZV<586>, ZPZV<3»; }; //
         template<> struct ConwayPolynomial<929, 9> { using ZPZ = aerobus::zpz<929>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<481>, ZPZV<199>, ZPZV<926»;</pre>
06038
         }; // NOLINT
               template<> struct ConwayPolynomial<937, 1> { using ZPZ = aerobus::zpz<937>; using type =
         POLYV<ZPZV<1>, ZPZV<932»; }; // NOLINT
06040
              template<> struct ConwayPolynomial<937, 2> { using ZPZ = aerobus::zpz<937>; using type =
         POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<937, 3> { using ZPZ = aerobus::zpz<937>; using type =
06041
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZV<32, ZPZY<32; }; // NOLINT template<> struct ConwayPolynomial<937, 4> { using ZPZ = aerobus::zpz<937>; using type =
06042
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<23>, ZPZV<585>, ZPZV<5»; }; // NOLINT
06043
               template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<932»; }; // NOLINT template<> struct ConwayPolynomial<937, 6> { using ZPZ = aerobus::zpz<937>; using type =
06044
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<794>, ZPZV<794>, ZPZV<934>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<937, 7> { using ZPZ = aerobus::zpz<937>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<932»; };
               template<> struct ConwayPolynomial<937, 8> { using ZPZ = aerobus::zpz<937>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<26>, ZPZV<53>, ZPZV<5»; };
         NOLINT
         template<> struct ConwayPolynomial<937, 9> { using ZPZ = aerobus::zpz<937>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<533>, ZPZV<483>, ZPZV<932»;
06047
         }; // NOLINT
               template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
06048
         POLYV<ZPZV<1>, ZPZV<939»; }; // NOLINT
06049
               template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
         POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
06050
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type =
06051
         \label{eq:polyv} \mbox{PDLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2*; }; \mbox{ }// \mbox{ NOLINT}
               template<> struct ConwayPolynomial<941, 5> { using ZPZ = aerobus::zpz<941>; using type =
06052
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<939»; }; // NOLINT
         template<> struct ConwayPolynomial<941, 6> { using ZPZ = aerobus::zpz<941>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<459>, ZPZV<694>, ZPZV<538>, ZPZV<2»; }; // NOLINT
06053
               template<> struct ConwayPolynomial<941, 7> { using ZPZ = aerobus::zpz<941>; using type
06054
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 8> { using ZPZ = aerobus::zpz<941>; using type =
06055
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<675>, ZPZV<675>, ZPZV<590>, ZPZV<2»; }; //
         NOLINT
06056
               template<> struct ConwayPolynomial<941, 9> { using ZPZ = aerobus::zpz<941>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<708>, ZPZV<708>, ZPZV<708>,
         }; // NOLINT
06057
               POLYV<ZPZV<1>, ZPZV<945»; }; // NOLINT
               template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2»; };
               template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 4> { using ZPZ = aerobus::zpz<947>; using type =
06060
                                                                                                // NOLINT
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<947, 5> { using ZPZ = aerobus::zpz<947>; using type =
06061
         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 =
06062
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<880>, ZPZV<787>, ZPZV<95>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<947, 7> { using ZPZ = aerobus::zpz<947>; using type =
06063
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<64>, ZPZV<945»; }; // NOLINT
              template<> struct ConwayPolynomial<947, 8> { using ZPZ = aerobus::zpz<947>; using type =
06064
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<845>, ZPZV<597>, ZPZV<581>, ZPZV<2»; }; //
06065
              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
06066
               template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
         POLYV<ZPZV<1>, ZPZV<950»; }; // NOLINT
               template<> struct ConwayPolynomial<953, 2> { using ZPZ = aerobus::zpz<953>; using type =
         POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3»; }; // NOLINT
06068
               template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950»; }; // NOLINT
template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<865>, ZPZV<3»; }; // NOLINT
06069
               template<> struct ConwayPolynomial<953, 5> { using ZPZ = aerobus::zpz<953>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<950»; }; // NOLINT
06071
               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
06072
              template<> struct ConwayPolynomial<953, 7> { using ZPZ = aerobus::zpz<953>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>; // NOLINT 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»; }; //
         template<> struct ConwayPolynomial<953, 9> { using ZPZ = aerobus::zpz<953>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<819>, ZPZV<316>, ZPZV<950»;</pre>
06074
         }; // NOLINT
               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 =
        POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5%; }; // NOLINT template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
06077
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
06078
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; }; // NOLINT
06079
               template<> struct ConwayPolynomial<967, 5> { using ZPZ = aerobus::zpz<967>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<962»; }; // NOLINT
06080
              template<> struct ConwayPolynomial<967, 6> { using ZPZ = aerobus::zpz<967>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<948>, ZPZV<831>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<967, 7> { using ZPZ = aerobus::zpz<967>; using type =
06081
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<962»; };
               template<> struct ConwayPolynomial<967, 8> { using ZPZ = aerobus::zpz<967>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<840>, ZPZV<502>, ZPZV<136>, ZPZV<5»; }; //
              template<> struct ConwayPolynomial<967, 9> { using ZPZ = aerobus::zpz<967>; using type =
06083
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<512>, ZPZV<783>, ZPZV<962»;
06084
              template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
         POLYV<ZPZV<1>, ZPZV<965»; }; // NOLINT
06085
               template<> struct ConwayPolynomial<971, 2> { using ZPZ = aerobus::zpz<971>; using type =
         POLYV<ZPZV<1>, ZPZV<970>, ZPZV<6»; }; // NOLINT
              template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
06086
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<965»; }; // NOLINT
               template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<527>, ZPZV<6»; }; // NOLINT
06088
              template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
          \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<14>, \verb"ZPZV<965"; \verb"}; $ // \verb"NOLINT" | NOLINT" | NOLINT | NOL
        template<> struct ConwayPolynomial<971, 60 { using ZPZ = aerobus::zpz<971>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<729>, ZPZV<718>, ZPZV<6»; }; // NOLINT
06089
               template<> struct ConwayPolynomial<971,
                                                                            7> { using ZPZ = aerobus::zpz<97</pre>
                                                                                                                               1>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<965»; }; // NOLI template<> struct ConwayPolynomial<971, 8> { using ZPZ = aerobus::zpz<971>; using type =
06091
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<281>, ZPZV<206>, ZPZV<6»; }; //
         NOLINT
              template<> struct ConwayPolynomial<971, 9> { using ZPZ = aerobus::zpz<971>; using type =
06092
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<805>, ZPZV<473>, ZPZV<965»;
         }; // NOLINT
06093
               template<> struct ConwayPolynomial<977, 1> { using ZPZ = aerobus::zpz<977>; using type =
         POLYV<ZPZV<1>, ZPZV<974»; }; // NOLINT
               template<> struct ConwayPolynomial<977, 2> { using ZPZ = aerobus::zpz<977>; using type =
06094
         POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<977, 3> { using ZPZ = aerobus::zpz<977>; using type =
06095
        POLYV<ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<974»; }; // NOLINT template<> struct ConwayPolynomial<977, 4> { using ZPZ = aerobus::zpz<977>; using type =
06096
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<800>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<977, 5> { using ZPZ = aerobus::zpz<977>; using type =
06097
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<01>, ZPZV<11>, ZPZV<974»; }; // NOLINT
```

```
06098
                      template<> struct ConwayPolynomial<977, 6> { using ZPZ = aerobus::zpz<977>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<729>, ZPZV<630>, ZPZV<753>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<977, 7> { using ZPZ = aerobus::zpz<977>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<974*; }; // NOLINT template<> struct ConwayPolynomial<977, 8> { using ZPZ = aerobus::zpz<977>; using type =
06100
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<855>, ZPZV<807>, ZPZV<77>, ZPZV<3»; }; //
                      template<> struct ConwayPolynomial<977, 9> { using ZPZ = aerobus::zpz<977>; using type
06101
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<740>, ZPZV<974»;
             }; // NOLINT
06102
                      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 =
06103
             POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; }; // NOLINT
06104
                     template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978»; ); // NOLINT
template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
06105
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<978»; }; // NOLINT
                      template<> struct ConwayPolynomial<983, 6> { using ZPZ = aerobus::zpz<983>; using type =
06107
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<849>, ZPZV<296>, ZPZV<228>, ZPZV<5»; }; // NOLINT
06108
                     template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
06109
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<5»; }; //
             template<> struct ConwayPolynomial<983, 9> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<85*>, ZPZV<85*>, ZPZV<87>, ZPZV<978*;</pre>
06110
             }; // NOLINT
                      template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
06111
             POLYV<ZPZV<1>, ZPZV<985»; }; // NOLINT
                      template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
06112
             POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; }; // NOLINT
06113
                      template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985»; }; // NOLINT template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6»; }; // NOLINT
06114
06115
                      template<> struct ConwayPolynomial<991, 5> { using ZPZ = aerobus::zpz<991>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<985»; }; // NOLINT
             template<> struct ConwayPolynomial<991, 6> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<637>, ZPZV<855>, ZPZV<278>, ZPZV<69*; }; // NOLINT
template<> struct ConwayPolynomial<991, 7> { using ZPZ = aerobus::zpz<991>; using type =
06116
06117
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      template<> struct ConwayPolynomial<991, 8> { using ZPZ = aerobus::zpz<991>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<941>, ZPZV<786>, ZPZV<234>, ZPZV<6»; }; //
             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<466>, ZPZV<266>, ZPZV<222>, ZPZV<985»;</pre>
06119
             }; // NOLINT
                      template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
             POLYV<ZPZV<1>, ZPZV<990»; }; // NOLINT
06121
                      template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
             POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
06122
             POLYY<ZPZY<1>, ZPZV<0>, ZPZV<2>, ZPZV<990»; }; // NOLINT template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7»; }; // NOLINT
                      template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
06124
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<990»; }; // NOLINT
                     template<> struct ConwayPolynomial<997, 6> { using ZPZ = aerobus::zpz<997>; using type =
06125
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<981>, ZPZV<260>, ZPZV<60>, ZPZV<69*, ZPZV<79*; }; // NOLINT template<> struct ConwayPolynomial<997, 7> { using ZPZ = aerobus::zpz<997>; using type
06126
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<990»; }; // NOLINT
06127
                     template<> struct ConwayPolynomial<997, 8> { using ZPZ = aerobus::zpz<997>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<934>, ZPZV<473>, ZPZV<241>, ZPZV<7»; };
             NOLINT
                     template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type =
06128
             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<39>, ZPZV<39>, ZPZV<732>, ZPZV<616>, ZPZV<990»;
              }; // NOLINT
06129 #endif // DO_NOT_DOCUMENT
06130 } // namespace aerobus
06131 #endif // AEROBUS_CONWAY_IMPORTS
06132
06133 #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\_

9.5 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<3>::eval(10) « '=' « aerobuslib::H3(10) « '\n';
        std::cout « aerobus::known_polynomials::hermite_phys<4>::eval(10) « '=' « aerobuslib::H3(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>
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;
}
```

192 Examples

#### 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
// \ {\tt TO GET optimal performances, modify cuda\_fpl6.h \ by \ adding \ \_CUDA\_FPl6\_CONSTEXPR\_ \ to \ line \ 5039 \ (version by the context of the context 
             12.6)
// Beforehand, you need to modify cuda_fp16.h by adding __CUDA_FP16_CONSTEXPR__ to line 5039 (version 12.6)
»»»> main
#include <cstdio>
#define WITH_CUDA_FP16
#include "../src/aerobus.h"
You may want to change int_type to aerobus::i32 (or i64) and float_type to float (resp. double)
using int_type = aerobus::i16;
using float_type = __half2;
constexpr size_t N = 1 « 26;
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 Expm1Degree<__half2> {
          static constexpr size_t val = 6;
template<>
struct Expm1Degree<__half> {
          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)));
} ;
template<>
struct GetRandT<__half> {
    static __half func(double min, double max) {
                  return __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) {
    «««< HEAD
       out[i] = f(f(f(f(f(f(f(f(f(f(f(f(in[i]))))))))));
       // fp16 FMA pipeline is quite wide so we need to feed it with a LOT of computations
       www> main
   }
#define cudaErrorCheck(ans) { gpuAssert((ans), __FILE__, __LINE__); }
inline void gpuAssert(cudaError_t code, const char *file, int line, bool abort=true)
   if (code != cudaSuccess)
      fprintf(stderr, "GPUassert: %s %s %d\n", cudaGetErrorString(code), file, line);\\
      if (abort) exit(code);
}
int main() {
    // configure CUDA devices
    int deviceCount;
    int device = -1;
    int maxProcCount = 0;
    cudaErrorCheck(cudaGetDeviceCount(&deviceCount));
    for(int i = 0; i < deviceCount; ++i) {</pre>
        cudaDeviceProp prop;
        cudaErrorCheck(cudaGetDeviceProperties(&prop, i));
        int procCount = prop.multiProcessorCount;
if(procCount > maxProcCount) {
            maxProcCount = procCount;
            device = i;
        }
    if(device == -1) {
        ::printf("CANNOT FIND CUDA CAPABLE DEVICE -- aborting\n");
        ::abort();
    cudaErrorCheck(cudaSetDevice(device));
    int blockSize; // The launch configurator returned block size int minGridSize; // The minimum grid size needed to achieve the
                    // maximum occupancy for a full device launch
    cudaErrorCheck(cudaOccupancyMaxPotentialBlockSize( &minGridSize, &blockSize, &run, 0, 0));
    ::printf("configure launch bounds to %d-%d\n", minGridSize, blockSize);
    // allocate and populate memory
    float_type *d_in, *d_out;
    cudaErrorCheck(cudaMalloc<float_type>(&d_in, N * sizeof(float_type)));
    cudaErrorCheck(cudaMalloc<float_type>(&d_out, N * sizeof(float_type)));
    \verb|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|cudaErrorCheck(cudaMemcpy(d_in, in, N * sizeof(float_type), cudaMemcpyHostToDevice));|
    // execute kernel and get memory back from device
    run«<minGridSize, blockSize»>(N, d_in, d_out);
    cudaErrorCheck(cudaPeekAtLastError());
    \verb|cudaErrorCheck(cudaMemopy(out, d_out, N * sizeof(float_type), cudaMemopyDeviceToHost));|\\
    cudaErrorCheck(cudaFree(d in)):
    cudaErrorCheck(cudaFree(d out));
// Example of generated SASS :
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875;
```

194 Examples

```
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625; HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625;
HFMA2.MMA R5, R6, R5, 0.5, 0.5;
HFMA2 R5, R6, R5, 1, 1;
HFMA2.MMA R5, R6, R5, RZ;
HFMA2 R7, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875;
HFMA2.MMA R7, R5, R7, 0.008331298828125, 0.008331298828125;
HFMA2 R7, R5, R7, 0.041656494140625, 0.041656494140625;
HFMA2.MMA R7, R5, R7, 0.1666259765625, 0.1666259765625; HFMA2 R7, R5, R7, 0.5, 0.5;
HFMA2.MMA R7, R5, R7, 1, 1;
HFMA2 R7, R5, R7, RZ.H0_H0;
HFMA2.MMA R5, R7, RZ, 0.0013885498046875, 0.0013885498046875;
HFMA2 R5, R7, R5, 0.008331298828125, 0.008331298828125;
HFMA2.MMA R5, R7, R5, 0.041656494140625, 0.041656494140625; HFMA2 R5, R7, R5, 0.1666259765625, 0.1666259765625;
HFMA2.MMA R5, R7, R5, 0.5, 0.5;
HFMA2 R5, R7, R5, 1, 1;
HFMA2.MMA R5, R7, R5, RZ;
HFMA2 R6, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875;
HFMA2.MMA R6, R5, R6, 0.008331298828125, 0.008331298828125; HFMA2 R6, R5, R6, 0.041656494140625, 0.041656494140625; HFMA2.MMA R6, R5, R6, 0.1666259765625, 0.1666259765625;
HFMA2 R6, R5, R6, 0.5, 0.5;
HFMA2.MMA R6, R5, R6, 1, 1;
HFMA2 R6, R5, R6, RZ.H0_H0;
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875 ;
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125; HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625;
HFMA2.MMA R5, R6, R5, 0.5, 0.5;
HFMA2 R5, R6, R5, 1, 1;
HFMA2.MMA R5, R6, R5, RZ;
HFMA2 R6, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875; HFMA2.MMA R6, R5, R6, 0.008331298828125, 0.008331298828125;
HFMA2 R6, R5, R6, 0.041656494140625, 0.041656494140625;
HFMA2.MMA R6, R5, R6, 0.1666259765625, 0.1666259765625;
HFMA2 R6, R5, R6, 0.5, 0.5;
HFMA2.MMA R6, R5, R6, 1, 1;
HFMA2 R6, R5, R6, RZ.H0_H0 ;
HFMA2 MMA R5, R6, R2, 0.0013885498046875, 0.0013885498046875; HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625;
HFMA2.MMA R5, R6, R5, 0.5, 0.5;
HFMA2 R5, R6, R5, 1, 1;
HFMA2.MMA R5, R6, R5, RZ; HFMA2 R6, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875;
HFMA2.MMA R6, R5, R6, 0.008331298828125, 0.008331298828125;
HFMA2 R6, R5, R6, 0.041656494140625, 0.041656494140625;
HFMA2.MMA R6, R5, R6, 0.1666259765625, 0.1666259765625;
HFMA2 R6, R5, R6, 0.5, 0.5;
HFMA2.MMA R6, R5, R6, 1, 1;
HFMA2 R6, R5, R6, RZ.H0_H0;
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875;
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625;
HFMA2.MMA R5, R6, R5, 0.5, 0.5;
HFMA2 R5, R6, R5, 1, 1;
HFMA2.MMA R5, R6, R5, RZ;
HFMA2 R6, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875;
HFMA2.MMA R6, R5, R6, 0.008331298828125, 0.008331298828125;
HFMA2 R6, R5, R6, 0.041656494140625, 0.041656494140625;
HFMA2.MMA R6, R5, R6, 0.1666259765625, 0.1666259765625;
HFMA2 R6, R5, R6, 0.5, 0.5;
HFMA2.MMA R6, R5, R6, 1, 1;
HFMA2 R6, R5, R6, RZ.H0_H0;
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875;
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625;
HFMA2.MMA R5, R6, R5, 0.5, 0.5;
HFMA2 R5, R6, R5, 1, 1;
HFMA2.MMA R6, R6, R5, RZ;
HFMA2 R5, R6, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875;
HFMA2.MMA R5, R6, R5, 0.008331298828125, 0.008331298828125; HFMA2 R5, R6, R5, 0.041656494140625, 0.041656494140625;
HFMA2.MMA R5, R6, R5, 0.1666259765625, 0.1666259765625;
HFMA2 R5, R6, R5, 0.5, 0.5;
HFMA2.MMA R7, R6, R5, 1, 1;
IADD3.X R5, R8, UR11, RZ, P0, !PT;
IADD3 R3, P0, R2, R3, RZ;
IADD3.X RO, RZ, RO, RZ, PO, !PT;
ISETP.GE.U32.AND PO, PT, R3, UR8, PT;
HFMA2 R7, R6, R7, RZ.H0_H0;
```

```
ISETP.GE.U32.AND.EX PO, PT, RO, UR9, PT, PO;
STG.E desc[UR6][R4.64], R7;
*/
```

#### 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;
}
```

#### 10.6 examples/make polynomial.cpp

```
How to build your own sequence of known polynomials, here Abel polynomials
```

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

196 Examples

```
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;
    return 0;
}
```

#### 10.7 examples/polynomials\_over\_finite\_field.cpp

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

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

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

int main() {
    // at this point, value_at_1 is an instanciation of zpz<2>::val
    using value_at_1 = P::template value_at_t<GF2::template inject_constant_t<1»;
    // here we get its value in an arithmetic type, here int32_t
    constexpr int32_t x = value_at_1::template get<int32_t>();
    // ensure that 1+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

See also

```
publication
// run with ./generate_comp_horner.sh in this directory
// that will compile and run this sample and plot all the generated data
#include "../src/aerobus.h"
using namespace aerobus; // NOLINT
constexpr size_t NB_POINTS = 400:
template<typename P, typename T, bool compensated>
DEVICE INLINED T eval(const T& x) {
    if constexpr (compensated) {
        return P::template compensated_eval<T>(x);
    } else {
        return P::template eval<T>(x);
template<typename T>
DEVICE 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) {</pre>
          ::fprintf(f, "%e %e %e\n", x, eval<P, T, compensated>(x), exact(x));
          x += step;
     ::fclose(f);
}
int main() {
           // (0.75 - x)^5 * (1 - x)^11
          using P = mul_t<
               pow_t<pq64, pq64::val<
                     typename q64::template inject_constant_t<-1>,
                     q64::val<i64::val<3>, i64::val<4>», 5>,
               pow_t<pq64. pq64::val<typename q64::template inject_constant_t<-1>, typename q64::one>, 11>
          >;
          vising FLOAT = double;
run<P, FLOAT, false>(0.68, 1.15, "plots/large_sample_horner.dat", &exact_large);
run<P, FLOAT, true>(0.68, 1.15, "plots/large_sample_comp_horner.dat", &exact_large);
          run<P, FLOAT, false>(0.74995, 0.75005, "plots/first_root_horner.dat", &exact_large);
run<P, FLOAT, true>(0.74995, 0.75005, "plots/first_root_comp_horner.dat", &exact_large);
          run<P, FLOAT, false>(0.9935, 1.0065, "plots/second_root_horner.dat", &exact_large);
run<P, FLOAT, true>(0.9935, 1.0065, "plots/second_root_comp_horner.dat", &exact_large);
           // (x - 1) ^ 3
          using P = make_int_polynomial_t<i32, 1, -3, 3, -1>;
          run<P, double, false>(1-0.00005, 1+0.00005, "plots/double.dat", &exact_small);
           run<P, float, true>(1-0.00005, 1+0.00005, "plots/float_comp.dat", &exact_small);
}
```

198 Examples

## Index

```
abs t
                                                              mulfractions t, 28
     aerobus, 21
                                                              pi64, 28
add t
                                                              PI fraction, 28
    aerobus, 21
                                                              pow t, 28
    aerobus::i32, 51
                                                              pq64, 28
    aerobus::i64, 56
                                                              q32, 28
    aerobus::polynomial < Ring >, 63
                                                              q64, 28
    aerobus::Quotient < Ring, X >, 69
                                                              sin, 28
    aerobus::zpz, 90
                                                              sinh, 29
                                                              SQRT2 fraction, 29
addfractions t
    aerobus, 22
                                                              SQRT3 fraction, 29
aerobus, 17
                                                              stirling 1 signed t, 29
    abs_t, 21
                                                              stirling_1_unsigned_t, 29
    add_t, 21
                                                              stirling_2_t, 30
    addfractions t, 22
                                                              sub t, 30
    aligned_malloc, 31
                                                              tan, 30
    alternate_t, 22
                                                              tanh, 30
    alternate_v, 32
                                                              taylor, 30
    asin, 22
                                                              vadd t, 31
    asinh, 22
                                                              vmul_t, 31
    atan, 23
                                                         aerobus::ContinuedFraction < a0 >, 42
    atanh, 23
                                                              type, 43
    bell t, 23
                                                              val, 43
    bernoulli t, 23
                                                         aerobus::ContinuedFraction < a0, rest... >, 43
    bernoulli v, 32
                                                              type, 44
    combination t, 23
                                                              val, 44
    combination v, 32
                                                         aerobus::ContinuedFraction < values >, 42
                                                         aerobus::ConwayPolynomial, 44
    cos, 24
    cosh, 24
                                                         aerobus::Embed< i32, i64 >, 45
    div t, 24
                                                              type, 46
    E fraction, 24
                                                         aerobus::Embed< polynomial< Small >, polynomial<
    embed_int_poly_in_fractions_t, 24
                                                                  Large > >, 46
    exp, 25
                                                              type, 46
    expm1, 25
                                                         aerobus::Embed< q32, q64 >, 47
    factorial t, 25
                                                              type, 47
    factorial_v, 32
                                                         aerobus::Embed< Quotient< Ring, X >, Ring >, 47
    field, 32
    fpq32, 25
                                                         aerobus::Embed< Ring, FractionField< Ring > >, 48
    fpq64, 25
                                                              type, 49
                                                         aerobus::Embed< Small, Large, E >, 45
    FractionField, 25
    gcd_t, 26
                                                         aerobus::Embed< zpz< x>, i32>, 49
    geometric sum, 26
                                                              type, 49
    Inp1, 26
                                                         aerobus::i32, 50
                                                              add t, 51
    make_frac_polynomial_t, 26
    make int polynomial t, 26
                                                              div t, 52
    make q32 t, 27
                                                              eq t, 52
    make_q64_t, 27
                                                              eq_v, 54
    makefraction_t, 27
                                                              gcd_t, 52
    mul t, 27
                                                              gt_t, 52
```

inject_constant_t, 52	aerobus::known_polynomials, 37
inject_ring_t, 53	hermite_kind, 37
inner_type, 53	physicist, 37
is_euclidean_domain, 54	probabilist, 37
is_field, 54	aerobus::polynomial < Ring >, 61
lt_t, 53	add_t, 63
mod_t, 53	derive_t, 63
mul_t, 53	div_t, 63
one, 53	eq_t, 63
pos_t, 53	gcd t, 64
pos_v, 54	gt_t, 64
sub_t, 54	inject_constant_t, 64
zero, 54	inject_ring_t, 64
aerobus::i32::val< x >, 76	is_euclidean_domain, 67
enclosing_type, 76	is_field, 67
get, 76	It_t, 65
is_zero_t, 76	mod_t, 65
to_string, 78	monomial_t, 65
v, 78	mul_t, 65
aerobus::i64, 54	one, 66
add_t, 56	pos_t, 66
div_t, 56	pos_v, 67
eq_t, 56	simplify_t, 66
eq_v, 59	sub_t, 66
gcd_t, 57	X, 66
gt_t, 57	zero, 66
gt_v, 59	aerobus::polynomial< Ring >::compensated_horner<
inject_constant_t, 57	arithmeticType, P >::EFTHorner< index,
inject_ring_t, 57	ghost >, 44
inner_type, 57	func, 44
is_euclidean_domain, 59	aerobus::polynomial< Ring >::compensated_horner<
is_field, 59	arithmeticType, P >::EFTHorner<-1, ghost >,
It_t, 57	45
It_v, 59	func, 45
mod_t, 58	aerobus::polynomial < Ring >::horner_reduction_t < P
	· · · — — —
mul_t, 58	>, 50
one, 58	aerobus::polynomial < Ring >::horner_reduction_t < P
pos_t, 58	>::inner< index, stop >, 60
pos_v, 59	type, 60
sub_t, 58	aerobus::polynomial< Ring >::horner_reduction_t< P
zero, 59	>::inner< stop, stop >, 60
aerobus::i64::val< x >, 78	type, 60
enclosing_type, 79	aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN $>$ , 85
get, 79	aN, 86
inner_type, 79	coeff_at_t, 86
is_zero_t, 79	compensated_eval, 87
to_string, 79	degree, 88
v, 79	enclosing_type, 87
aerobus::internal, 33	eval, 87
index_sequence_reverse, 36	is_zero_t, 87
is_instantiation_of_v, 37	is_zero_v, 88
make_index_sequence_reverse, 36	ring_type, 87
type_at_t, 36	strip, 87
aerobus::is_prime< n >, 61	to_string, 88
value, 61	value_at_t, 87
aerobus::IsEuclideanDomain, 39	
	aerobus::polynomial < Ring >::val < coeffN >::coeff_at <
aerobus::IsField, 39	index, E >, 41
aerobus::IsRing, 39	aerobus::polynomial < Ring >::val < coeffN >::coeff_at <

```
index, std::enable_if_t<(index< 0 | | index >
                                                               add_t, 90
                                                               div t, 90
          0)>>, 41
     type, 41
                                                               eq_t, 90
aerobus::polynomial < Ring >::val < coeffN >::coeff_at <
                                                               eq_v, 93
          index, std::enable_if_t<(index==0)>>, 41
                                                               gcd_t, 90
                                                               gt t, 91
aerobus::polynomial< Ring >::val< coeffN, coeffs >,
                                                               gt_v, 93
                                                               inject_constant_t, 91
          80
     aN, 81
                                                               inner type, 91
     coeff_at_t, 81
                                                               is euclidean domain, 93
     compensated_eval, 82
                                                               is field, 93
     degree, 83
                                                               lt_t, 91
     enclosing_type, 81
                                                               It_v, 93
     eval, 82
                                                               mod_t, 91
                                                               mul_t, 92
     is_zero_t, 81
     is_zero_v, 83
                                                               one, 92
     ring type, 81
                                                               pos t, 92
     strip, 81
                                                               pos v, 93
     to_string, 82
                                                               sub_t, 92
     value_at_t, 81
                                                               zero, 92
aerobus::Quotient < Ring, X >, 68
                                                          aerobus::zpz<p>::val<math><x>, 84
     add t, 69
                                                               enclosing type, 84
                                                               get, 85
     div_t, 69
     eq_t, 69
                                                               is_zero_t, 84
     eq_v, 71
                                                               is zero v, 85
     inject_constant_t, 69
                                                               to_string, 85
     inject_ring_t, 70
                                                               v, 85
     is euclidean domain, 71
                                                          aligned malloc
     mod t, 70
                                                               aerobus, 31
                                                          alternate_t
     mul t, 70
     one, 70
                                                               aerobus, 22
     pos_t, 71
                                                          alternate_v
     pos v, 71
                                                               aerobus, 32
                                                          aN
     zero, 71
aerobus::Quotient < Ring, X >::val < V >, 83
                                                               aerobus::polynomial < Ring >::val < coeffN >, 86
     raw_t, 83
                                                               aerobus::polynomial< Ring >::val< coeffN, coeffs
     type, 83
                                                                    >, 81
                                                          asin
aerobus::type_list< Ts >, 72
     at, 73
                                                               aerobus, 22
     concat, 73
                                                          asinh
     insert, 74
                                                               aerobus, 22
     length, 74
                                                          at
     push_back, 74
                                                               aerobus::type_list< Ts >, 73
     push front, 74
                                                          atan
     remove, 74
                                                               aerobus, 23
aerobus::type_list< Ts >::pop_front, 67
                                                          atanh
     tail, 67
                                                               aerobus, 23
     type, 67
                                                          bell t
aerobus::type_list< Ts >::split< index >, 72
                                                               aerobus, 23
     head, 72
                                                          bernoulli t
     tail, 72
                                                               aerobus, 23
aerobus::type list<>, 75
                                                          bernoulli v
     concat, 75
                                                               aerobus, 32
     insert, 75
     length, 75
                                                          coeff at t
     push_back, 75
                                                               aerobus::polynomial < Ring >::val < coeffN >, 86
     push_front, 75
                                                               aerobus::polynomial< Ring >::val< coeffN, coeffs
aerobus::zpz< p>, 88
                                                                    >, 81
```

```
combination_t
                                                            aerobus, 25
     aerobus, 23
                                                       factorial_t
combination v
                                                            aerobus, 25
    aerobus, 32
                                                       factorial v
compensated eval
                                                            aerobus, 32
     aerobus::polynomial< Ring >::val< coeffN >, 87
                                                       field
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                            aerobus, 32
          >, 82
                                                       fpq32
concat
                                                            aerobus, 25
     aerobus::type_list< Ts >, 73
                                                       fpq64
     aerobus::type_list<>, 75
                                                            aerobus, 25
cos
                                                       FractionField
     aerobus, 24
                                                            aerobus, 25
cosh
     aerobus, 24
                                                       func
                                                            aerobus::polynomial < Ring >::compensated horner <
                                                                 arithmeticType, P >::EFTHorner< index,
degree
     aerobus::polynomial < Ring >::val < coeffN >, 88
                                                                 ghost >, 44
                                                            aerobus::polynomial < Ring >::compensated_horner <
     aerobus::polynomial < Ring >::val < coeffN, coeffs
                                                                 arithmeticType, P >::EFTHorner <- 1, ghost >,
derive t
                                                                 45
     aerobus::polynomial < Ring >, 63
                                                       gcd t
div t
                                                            aerobus, 26
     aerobus, 24
                                                            aerobus::i32, 52
     aerobus::i32, 52
                                                            aerobus::i64, 57
     aerobus::i64, 56
                                                            aerobus::polynomial < Ring >, 64
     aerobus::polynomial < Ring >, 63
                                                            aerobus::zpz< p>, 90
     aerobus::Quotient < Ring, X >, 69
                                                        geometric_sum
     aerobus::zpz, 90
                                                            aerobus, 26
E fraction
                                                       get
     aerobus, 24
                                                            aerobus::i32::val< x >, 76
embed_int_poly_in_fractions_t
                                                            aerobus::i64::val < x > 79
    aerobus, 24
                                                            aerobus::zpz ::val < x >, 85
enclosing type
                                                       gt_t
     aerobus::i32::val< x >, 76
                                                            aerobus::i32, 52
     aerobus::i64::val < x > 79
                                                            aerobus::i64, 57
     aerobus::polynomial < Ring >::val < coeffN >, 87
                                                            aerobus::polynomial < Ring >, 64
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                            aerobus::zpz , 91
         >, 81
                                                       gt_v
     aerobus::zpz ::val < x >, 84
                                                            aerobus::i64, 59
eq t
                                                            aerobus::zpz , 93
     aerobus::i32, 52
                                                       head
     aerobus::i64, 56
                                                            aerobus::type list< Ts >::split< index >, 72
     aerobus::polynomial < Ring >, 63
     aerobus::Quotient < Ring, X >, 69
                                                       hermite kind
                                                            aerobus::known_polynomials, 37
     aerobus::zpz, 90
eq_v
                                                       index_sequence_reverse
     aerobus::i32, 54
                                                            aerobus::internal, 36
     aerobus::i64, 59
                                                       inject constant t
     aerobus::Quotient< Ring, X >, 71
                                                            aerobus::i32, 52
     aerobus::zpz , 93
                                                            aerobus::i64, 57
eval
                                                            aerobus::polynomial < Ring >, 64
     aerobus::polynomial< Ring >::val< coeffN >, 87
                                                            aerobus::Quotient < Ring, X >, 69
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                            aerobus::zpz, 91
         >, 82
                                                       inject_ring_t
exp
                                                            aerobus::i32, 53
    aerobus, 25
                                                            aerobus::i64, 57
expm1
```

parahua::nalunamial < Ping > 64	make a64 t
aerobus::polynomial < Ring >, 64	make_q64_t
aerobus::Quotient< Ring, X >, 70	aerobus, 27
inner_type	makefraction_t
aerobus::i32, 53	aerobus, 27
aerobus::i64, 57	mod_t
aerobus:: $i64::val < x > 79$	aerobus::i32, 53
aerobus::zpz, 91	aerobus::i64, 58
insert	aerobus::polynomial $<$ Ring $>$ , 65
aerobus::type_list< Ts >, 74	aerobus::Quotient $<$ Ring, $X>$ , 70
aerobus::type_list<>, 75	aerobus:: $zpz $ , 91
Introduction, 1	monomial_t
is_euclidean_domain	aerobus::polynomial < Ring >, 65
aerobus::i32, 54	mul_t
aerobus::i64, 59	aerobus, 27
aerobus::polynomial< Ring >, 67	aerobus::i32, 53
aerobus::Quotient< Ring, X >, 71	aerobus::i64, 58
aerobus:: $zpz  93$	aerobus::polynomial < Ring >, 65
is field	aerobus::Quotient< Ring, X >, 70
<del>-</del>	
aerobus::i32, 54	aerobus::zpz, 92
aerobus::i64, 59	mulfractions_t
aerobus::polynomial< Ring >, 67	aerobus, 28
aerobus:: $zpz $ , 93	
is_instantiation_of_v	one
aerobus::internal, 37	aerobus::i32, 53
is_zero_t	aerobus::i64, 58
aerobus::i32::val < $x >$ , 76	aerobus::polynomial $<$ Ring $>$ , 66
aerobus::i64::val $< x >$ , 79	aerobus::Quotient $<$ Ring, $X>$ , 70
aerobus::polynomial < Ring >::val < coeffN >, 87	aerobus:: $zpz $ , 92
aerobus::polynomial< Ring >::val< coeffN, coeffs	
>, 81	physicist
aerobus::zpz::val< x >, 84	aerobus::known_polynomials, 37
is zero v	pi64
aerobus::polynomial< Ring >::val< coeffN >, 88	aerobus, 28
aerobus::polynomial< Ring >::val< coeffN, coeffs	PI fraction
	aerobus, 28
>, 83	pos_t
aerobus::zpz $<$ p $>$ ::val $<$ x $>$ , 85	aerobus::i32, 53
langth	aerobus::i64, 58
length	aerobus::polynomial< Ring >, 66
aerobus::type_list< Ts >, 74	
aerobus::type_list<>, 75	aerobus::Quotient< Ring, X >, 71
Inp1	aerobus::zpz, 92
aerobus, 26	pos_v
lt_t	aerobus::i32, 54
aerobus::i32, 53	aerobus::i64, 59
aerobus::i64, 57	aerobus::polynomial $<$ Ring $>$ , 67
aerobus::polynomial < Ring >, 65	aerobus::Quotient $<$ Ring, $X>$ , 71
aerobus:: $zpz $ , 91	aerobus:: $zpz $ , 93
lt_v	pow_t
_ aerobus::i64, 59	aerobus, 28
aerobus::zpz, 93	pq64
4010040.12PZ	aerobus, 28
make free networmiel t	probabilist
make irac polynomiai i	aerobus::known_polynomials, 37
make_frac_polynomial_t	· ·
aerobus, 26	nush hack
aerobus, 26 make_index_sequence_reverse	push_back
aerobus, 26 make_index_sequence_reverse aerobus::internal, 36	aerobus::type_list< Ts >, 74
aerobus, 26 make_index_sequence_reverse aerobus::internal, 36 make_int_polynomial_t	aerobus::type_list< Ts >, 74 aerobus::type_list<>, 75
aerobus, 26 make_index_sequence_reverse aerobus::internal, 36 make_int_polynomial_t aerobus, 26	aerobus::type_list< Ts >, 74 aerobus::type_list<>, 75 push_front
aerobus, 26 make_index_sequence_reverse aerobus::internal, 36 make_int_polynomial_t	aerobus::type_list< Ts >, 74 aerobus::type_list<>, 75

q32	aerobus::zpz $<$ p $>$ ::val $<$ x $>$ , 85
aerobus, 28	type
q64	aerobus::ContinuedFraction< a0 >, 43
aerobus, 28	aerobus::ContinuedFraction< a0, rest >, 44
ways t	aerobus::Embed< i32, i64 >, 46
raw_t	aerobus::Embed< polynomial< Small >,
aerobus::Quotient< Ring, X >::val< V >, 83 README.md, 95	polynomial < Large > >, 46
remove	aerobus::Embed < q32, q64 >, 47
aerobus::type_list< Ts >, 74	aerobus::Embed< Quotient< Ring, X >, Ring >, 48
ring_type	aerobus::Embed< Ring, FractionField< Ring > >,
aerobus::polynomial< Ring >::val< coeffN >, 87	49
aerobus::polynomial < Ring >::val < coeffN, coeffs	aerobus::Embed $<$ zpz $<$ x $>$ , i32 $>$ , 49
>, 81	aerobus::polynomial< Ring >::horner_reduction_t<
	P >::inner< index, stop >, 60
simplify_t	aerobus::polynomial< Ring >::horner_reduction_t<
aerobus::polynomial < Ring >, 66	P >::inner< stop, stop >, 60
sin	aerobus::polynomial< Ring >::val< coeffN
aerobus, 28	>::coeff_at< index, std::enable_if_t<(index<
sinh	$0 \mid   \text{index} > 0) > >, 41$
aerobus, 29	aerobus::polynomial< Ring >::val< coeffN
SQRT2_fraction aerobus, 29	>::coeff_at< index, std::enable_if_t<(index==0)>
SQRT3_fraction	>, 42
aerobus, 29	aerobus::Quotient< Ring, X >::val< V >, 83
src/aerobus.h, 95	aerobus::type_list< Ts >::pop_front, 67
src/examples.h, 188	type_at_t
stirling_1_signed_t	aerobus::internal, 36
aerobus, 29	V
stirling_1_unsigned_t	aerobus::i32::val< x >, 78
aerobus, 29	aerobus::i64::val< x >, 79
stirling_2_t	aerobus::zpz $<$ p $>$ ::val $<$ x $>$ , 85
aerobus, 30	vadd_t
strip	aerobus, 31
aerobus::polynomial< Ring >::val< coeffN >, 87	val
aerobus::polynomial< Ring >::val< coeffN, coeffs	aerobus::ContinuedFraction< a0 >, 43
>, 81	aerobus::ContinuedFraction< a0, rest >, 44
sub_t	value
aerobus, 30 aerobus::i32, 54	aerobus::is_prime< n >, 61
aerobus::i64, 58	value_at_t
aerobus::polynomial< Ring >, 66	aerobus::polynomial< Ring >::val< coeffN >, 87 aerobus::polynomial< Ring >::val< coeffN, coeffs
aerobus::zpz, 92	>, 81
чения при	vmul t
tail	aerobus, 31
aerobus::type_list< Ts >::pop_front, 67	4010040, 01
aerobus::type_list< Ts >::split< index >, 72	X
tan	aerobus::polynomial < Ring >, 66
aerobus, 30	
tanh	zero
aerobus, 30	aerobus::i32, 54
taylor	aerobus::i64, 59
aerobus, 30	aerobus::polynomial < Ring >, 66
to_string aerobus::i32::val< x >, 78	aerobus::Quotient< Ring, X >, 71 aerobus::zpz, 92
aerobus::i64::val $< x >$ , 79	αοιουασερε ν ν, σε
aerobus::polynomial< Ring >::val< coeffN >, 88	
aerobus::polynomial< Ring >::val< coeffN, coeffs	
>, 82	