## Aerobus

v1.2

Generated by Doxygen 1.9.8

1	Introduction	1
	1.1 HOW TO	1
	1.1.1 Unit Test	2
	1.1.2 Benchmarks	2
	1.2 Structures	2
	1.2.1 Predefined discrete euclidean domains	2
	1.2.2 Polynomials	3
	1.2.3 Known polynomials	3
	1.2.4 Conway polynomials	3
	1.2.5 Taylor series	4
	1.3 Operations	5
	1.3.1 Field of fractions	5
	1.3.2 Quotient	6
	1.4 Misc	6
	1.4.1 Continued Fractions	6
	1.5 CUDA	6
•	Manager Indian	_
2	Namespace Index	7
	2.1 Namespace List	7
3	Concept Index	9
	3.1 Concepts	9
4	Class Index	11
	4.1 Class List	11
5	File Index	13
	5.1 File List	13
6	Namespace Documentation	15
	6.1 aerobus Namespace Reference	15
	6.1.1 Detailed Description	19
	6.1.2 Typedef Documentation	20
	6.1.2.1 abs_t	20
	6.1.2.2 add_t	20
	6.1.2.3 addfractions_t	20
	6.1.2.4 alternate_t	20
	6.1.2.5 asin	21
	6.1.2.6 asinh	21
	6.1.2.7 atan	21
	6.1.2.8 atanh	21
	6.1.2.9 bell_t	23
	6.1.2.10 bernoulli_t	23
	6.1.2.11 combination_t	23

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

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

8.5 aerobus::ContinuedFraction< a0 > Struct Template Reference	45
8.5.1 Detailed Description	45
8.5.2 Member Typedef Documentation	45
8.5.2.1 type	45
8.5.3 Member Data Documentation	46
8.5.3.1 val	46
8.6 aerobus::ContinuedFraction< a0, rest > Struct Template Reference	46
8.6.1 Detailed Description	46
8.6.2 Member Typedef Documentation	47
8.6.2.1 type	47
8.6.3 Member Data Documentation	47
8.6.3.1 val	47
8.7 aerobus::ConwayPolynomial Struct Reference	47
8.8 aerobus::Embed $<$ Small, Large, E $>$ Struct Template Reference	47
8.8.1 Detailed Description	47
8.9 aerobus::Embed< i32, i64 > Struct Reference	48
8.9.1 Detailed Description	48
8.9.2 Member Typedef Documentation	48
8.9.2.1 type	48
$8.10 \; aerobus :: Embed < polynomial < Small >, polynomial < Large > > Struct \; Template \; Reference  .  .  . \\$	49
8.10.1 Detailed Description	49
8.10.2 Member Typedef Documentation	49
8.10.2.1 type	49
8.11 aerobus::Embed < q32, q64 > Struct Reference	50
8.11.1 Detailed Description	50
8.11.2 Member Typedef Documentation	50
8.11.2.1 type	50
8.12 aerobus::Embed < Quotient < Ring, X >, Ring > Struct Template Reference	50
8.12.1 Detailed Description	51
8.12.2 Member Typedef Documentation	51
8.12.2.1 type	51
8.13 aerobus::Embed< Ring, FractionField< Ring >> Struct Template Reference	51
8.13.1 Detailed Description	51
8.13.2 Member Typedef Documentation	52
8.13.2.1 type	52
8.14 aerobus::Embed< zpz< x >, i32 > Struct Template Reference $\dots \dots \dots \dots \dots \dots \dots$	52
8.14.1 Detailed Description	52
8.14.2 Member Typedef Documentation	53
8.14.2.1 type	53
$8.15 \ aerobus::polynomial < Ring > ::horner\_reduction\_t < P > Struct \ Template \ Reference \ \ . \ \ . \ \ . \ \ .$	53
8.15.1 Detailed Description	53
8.16 aerobus::i32 Struct Reference	54

. 55
. 55
. 55
. 55
. 55
. 56
. 56
. 56
. 56
. 57
. 57
. 57
. 57
. 57
. 58
. 58
. 58
. 58
. 58
. 58
. 59
. 59
. 59
. 61
. 61
. 61
. 61
. 61
. 61
. 62
. 62
. 62
. 62
. 63
. 63
. 63
. 63
. 63
. 64
. 64
. 64
. 64

8.17.3.2 gt_v	64
8.17.3.3 is_euclidean_domain	65
8.17.3.4 is_field	65
8.17.3.5 lt_v	65
8.17.3.6 pos_v	65
$8.18 \ aerobus::polynomial < Ring > ::horner\_reduction\_t < P > ::inner < index, stop > Struct \ Template \ Refusion = (Applied to the property of the proper$	
erence	65
8.18.1 Member Typedef Documentation	66
8.18.1.1 type	66
$8.19\;aerobus::polynomial < Ring > ::horner\_reduction\_t < P > ::inner < stop, \; stop > Struct \; Template \; Refusion   Part   Pa$	
erence	66
8.19.1 Member Typedef Documentation	
8.19.1.1 type	
8.20 aerobus::is_prime < n > Struct Template Reference	67
8.20.1 Detailed Description	67
8.20.2 Member Data Documentation	67
8.20.2.1 value	67
8.21 aerobus::polynomial < Ring > Struct Template Reference	67
8.21.1 Detailed Description	69
8.21.2 Member Typedef Documentation	69
8.21.2.1 add_t	69
8.21.2.2 derive_t	69
8.21.2.3 div_t	70
8.21.2.4 eq_t	70
8.21.2.5 gcd_t	70
8.21.2.6 gt t	
8.21.2.7 inject_constant_t	
8.21.2.8 inject_ring_t	
8.21.2.9 lt_t	71
8.21.2.10 mod t	
8.21.2.11 monomial_t	72
8.21.2.12 mul_t	72
8.21.2.13 one	72
8.21.2.14 pos_t	72
8.21.2.15 simplify_t	74
8.21.2.16 sub_t	74
8.21.2.17 X	74
8.21.2.18 zero	74 74
8.21.3 Member Data Documentation	75 75
8.21.3.1 is_euclidean_domain	75
8.21.3.2 is_field	75
8.21.3.3 pos_v	75
8.22 aerobus: type_list< Ts >::non_front Struct Reference	75

8.22.1 Detailed Description	7
8.22.2 Member Typedef Documentation	70
8.22.2.1 tail	70
8.22.2.2 type	70
8.23 aerobus::Quotient $<$ Ring, X $>$ Struct Template Reference	70
8.23.1 Detailed Description	7
8.23.2 Member Typedef Documentation	7
8.23.2.1 add_t	7
8.23.2.2 div_t	78
8.23.2.3 eq_t	78
8.23.2.4 inject_constant_t	78
8.23.2.5 inject_ring_t	79
8.23.2.6 mod_t	79
8.23.2.7 mul_t	79
8.23.2.8 one	79
8.23.2.9 pos_t	8
8.23.2.10 zero	8
8.23.3 Member Data Documentation	8
8.23.3.1 eq_v	8
8.23.3.2 is_euclidean_domain	8
8.23.3.3 pos_v	8
8.23.3.3 pos_v	
• =	8
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference	8
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference	8 8
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference	8 8 8
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference	8 8 8 8
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference	8 8 8
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference	
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference  8.24.1 Detailed Description  8.24.2 Member Typedef Documentation  8.24.2.1 head  8.24.2.2 tail  8.25 aerobus::type_list< Ts > Struct Template Reference  8.25.1 Detailed Description	
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference  8.24.1 Detailed Description  8.24.2 Member Typedef Documentation  8.24.2.1 head  8.24.2.2 tail  8.25 aerobus::type_list< Ts > Struct Template Reference  8.25.1 Detailed Description  8.25.2 Member Typedef Documentation	
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference  8.24.1 Detailed Description  8.24.2 Member Typedef Documentation  8.24.2.1 head  8.24.2.2 tail  8.25 aerobus::type_list< Ts > Struct Template Reference  8.25.1 Detailed Description  8.25.2 Member Typedef Documentation  8.25.2.1 at .	
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference  8.24.1 Detailed Description  8.24.2 Member Typedef Documentation  8.24.2.1 head  8.24.2.2 tail  8.25 aerobus::type_list< Ts > Struct Template Reference  8.25.1 Detailed Description  8.25.2 Member Typedef Documentation  8.25.2.1 at  8.25.2.2 concat	
8.24 aerobus::type_list < Ts >::split < index > Struct Template Reference 8.24.1 Detailed Description 8.24.2 Member Typedef Documentation 8.24.2.1 head 8.24.2.2 tail 8.25 aerobus::type_list < Ts > Struct Template Reference 8.25.1 Detailed Description 8.25.2 Member Typedef Documentation 8.25.2.1 at 8.25.2.2 concat 8.25.2.3 insert	
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference  8.24.1 Detailed Description  8.24.2 Member Typedef Documentation  8.24.2.1 head  8.24.2.2 tail  8.25 aerobus::type_list< Ts > Struct Template Reference  8.25.1 Detailed Description  8.25.2 Member Typedef Documentation  8.25.2.1 at  8.25.2.2 concat  8.25.2.3 insert  8.25.2.4 push_back	
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference  8.24.1 Detailed Description  8.24.2 Member Typedef Documentation  8.24.2.1 head  8.24.2.2 tail  8.25 aerobus::type_list< Ts > Struct Template Reference  8.25.1 Detailed Description  8.25.2 Member Typedef Documentation  8.25.2.1 at  8.25.2.2 concat  8.25.2.3 insert  8.25.2.4 push_back  8.25.2.5 push_front	
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference  8.24.1 Detailed Description  8.24.2 Member Typedef Documentation  8.24.2.1 head  8.24.2.2 tail  8.25 aerobus::type_list< Ts > Struct Template Reference  8.25.1 Detailed Description  8.25.2 Member Typedef Documentation  8.25.2.1 at  8.25.2.2 concat  8.25.2.3 insert  8.25.2.4 push_back  8.25.2.5 push_front  8.25.2.6 remove	
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference  8.24.1 Detailed Description  8.24.2 Member Typedef Documentation  8.24.2.1 head  8.24.2.2 tail  8.25 aerobus::type_list< Ts > Struct Template Reference  8.25.1 Detailed Description  8.25.2 Member Typedef Documentation  8.25.2.1 at  8.25.2.2 concat  8.25.2.3 insert  8.25.2.3 insert  8.25.2.4 push_back  8.25.2.5 push_front  8.25.2.6 remove  8.25.3 Member Data Documentation	
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference	
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference 8.24.1 Detailed Description 8.24.2 Member Typedef Documentation 8.24.2.1 head 8.24.2.2 tail 8.25 aerobus::type_list< Ts > Struct Template Reference 8.25.1 Detailed Description 8.25.2 Member Typedef Documentation 8.25.2.1 at 8.25.2.2 concat 8.25.2.3 insert 8.25.2.3 insert 8.25.2.4 push_back 8.25.2.5 push_front 8.25.2.6 remove 8.25.3 Member Data Documentation 8.25.3.1 length 8.26 aerobus::type_list<> Struct Reference	
8.24 aerobus::type_list< Ts >::split< index > Struct Template Reference 8.24.1 Detailed Description 8.24.2 Member Typedef Documentation 8.24.2.1 head 8.24.2.2 tail 8.25 aerobus::type_list< Ts > Struct Template Reference 8.25.1 Detailed Description 8.25.2 Member Typedef Documentation 8.25.2.1 at 8.25.2.2 concat 8.25.2.3 insert 8.25.2.4 push_back 8.25.2.5 push_front 8.25.2.6 remove 8.25.3 Member Data Documentation 8.25.3.1 length 8.26 aerobus::type_list<> Struct Reference 8.26.1 Detailed Description	

8.26.2.3 push_back	85
8.26.2.4 push_front	85
8.26.3 Member Data Documentation	86
8.26.3.1 length	86
8.27 aerobus::i32::val $<$ x $>$ Struct Template Reference	86
8.27.1 Detailed Description	86
8.27.2 Member Typedef Documentation	87
8.27.2.1 enclosing_type	87
8.27.2.2 is_zero_t	87
8.27.3 Member Function Documentation	87
8.27.3.1 get()	87
8.27.3.2 to_string()	87
8.27.4 Member Data Documentation	87
8.27.4.1 v	87
8.28 aerobus::i64::val $<$ x $>$ Struct Template Reference	88
8.28.1 Detailed Description	88
8.28.2 Member Typedef Documentation	89
8.28.2.1 enclosing_type	89
8.28.2.2 inner_type	89
8.28.2.3 is_zero_t	89
8.28.3 Member Function Documentation	89
8.28.3.1 get()	89
8.28.3.2 to_string()	89
8.28.4 Member Data Documentation	90
8.28.4.1 v	90
8.29 aerobus::polynomial< Ring >::val< coeffN, coeffs > Struct Template Reference	90
8.29.1 Detailed Description	91
8.29.2 Member Typedef Documentation	91
8.29.2.1 aN	91
8.29.2.2 coeff_at_t	91
8.29.2.3 enclosing_type	92
8.29.2.4 is_zero_t	92
8.29.2.5 ring_type	92
8.29.2.6 strip	92
8.29.2.7 value_at_t	92
8.29.3 Member Function Documentation	92
8.29.3.1 eval()	92
8.29.3.2 to_string()	93
8.29.4 Member Data Documentation	93
8.29.4.1 degree	93
8.29.4.2 is_zero_v	93
8.30 aerobus::Quotient < Ring, X >::val < V > Struct Template Reference	94

8.30.1 Detailed Description	94
8.30.2 Member Typedef Documentation	94
8.30.2.1 raw_t	94
8.30.2.2 type	94
8.31 aerobus::zpz::val< x > Struct Template Reference	94
8.31.1 Detailed Description	95
8.31.2 Member Typedef Documentation	95
8.31.2.1 enclosing_type	95
8.31.2.2 is_zero_t	96
8.31.3 Member Function Documentation	96
8.31.3.1 get()	96
8.31.3.2 to_string()	96
8.31.4 Member Data Documentation	96
8.31.4.1 is_zero_v	96
8.31.4.2 v	97
8.32 aerobus::polynomial< Ring >::val< coeffN > Struct Template Reference	97
8.32.1 Detailed Description	98
8.32.2 Member Typedef Documentation	99
8.32.2.1 aN	99
8.32.2.2 coeff_at_t	99
8.32.2.3 enclosing_type	99
8.32.2.4 is_zero_t	99
8.32.2.5 ring_type	99
8.32.2.6 strip	99
8.32.2.7 value_at_t	00
8.32.3 Member Function Documentation	00
8.32.3.1 eval()	00
8.32.3.2 to_string()	00
8.32.4 Member Data Documentation	00
8.32.4.1 degree	00
8.32.4.2 is_zero_v	00
8.33 aerobus::zpz Struct Template Reference	01
8.33.1 Detailed Description	02
8.33.2 Member Typedef Documentation	02
8.33.2.1 add_t	02
8.33.2.2 div_t	03
8.33.2.3 eq_t	03
8.33.2.4 gcd_t	03
8.33.2.5 gt_t	04
8.33.2.6 inject_constant_t	04
8.33.2.7 inner_type	04
8.33.2.8 lt_t	04

Index

8.33.2.9 mod_t	105
8.33.2.10 mul_t	105
8.33.2.11 one	105
8.33.2.12 pos_t	105
8.33.2.13 sub_t	106
8.33.2.14 zero	106
8.33.3 Member Data Documentation	106
8.33.3.1 eq_v	106
8.33.3.2 gt_v	106
8.33.3.3 is_euclidean_domain	107
8.33.3.4 is_field	107
8.33.3.5 lt_v	107
8.33.3.6 pos_v	107
9 File Documentation	109
9.1 README.md File Reference	109
9.2 src/aerobus.h File Reference	109
9.3 aerobus.h	109
9.4 src/examples.h File Reference	200
9.5 examples.h	200
10 Examples	201
10.1 examples/hermite.cpp	201
10.2 examples/custom_taylor.cpp	201
10.3 examples/fp16.cu	
10.4 examples/continued_fractions.cpp	202
10.5 examples/modular_arithmetic.cpp	202
10.6 examples/make_polynomial.cpp	203
10.7 examples/polynomials_over_finite_field.cpp	

205

### 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 aerobus_benchmarks
./aerobus_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
- sub\_t : a type (specialization of val), representing subtraction between two values
- mul\_t : a type (specialization of val), representing multiplication between two values
- div\_t : a type (specialization of val), representing division between two values
- mod\_t : a type (specialization of val), representing modulus between two values

and the following "elements":

- one : the neutral element for multiplication, val<1>
- zero : the neutral element for addition, val < 0>

1.2 Structures 3

#### 1.2.2 Polynomials

Aerobus defines polynomials as a variadic template structure, with coefficient in an arbitrary discrete euclidean domain. As i32 or i64, they are given same operations and elements, which make them a euclidean domain by themselves. Similarly, aerobus::polynomial represents the algebraic structure, actual values are in aerobus::polynomial::val.

```
In addition, values have an evaluation function: template<typename valueRing> static constexpr valueRing eval(const valueRing& x) \{\ldots\}
```

Which can be used at compile time (constexpr evaluation) or runtime.

#### 1.2.3 Known polynomials

```
Aerobus predefines some well known families of polynomials, such as Hermite or Bernstein: using B23 = aerobus::known_polynomials::bernstein<2, 3>; // 3X^2(1-X) constexpr float x = B32::eval(2.0F); // -12
```

They have their coefficients either in aerobus::i64 or aerobus::q64. Complete list is (but is meant to be extended):

- chebyshev\_T
- chebyshev\_U
- · laquerre
- hermite\_prob
- hermite\_phys
- bernstein
- legendre
- bernoulli

#### 1.2.4 Conway polynomials

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

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

4 Introduction

#### 1.2.5 Taylor series

Aerobus provides definition for Taylor expansion of known functions. They are all templates in two parameters, degree of expansion ( $size\_t$ ) and Integers (typename). Coefficients then live in  $Fraction \leftarrow Field < Integers >$ .

#### They can be used and evaluated:

```
using namespace aerobus;
using aero_atanh = atanh<i64, 6>;
constexpr float val = aero_atanh::eval(0.1F); // approximation of arctanh(0.1) using taylor expansion of
    degree 6
```

#### Exposed functions are:

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

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

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

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

1.3 Operations 5

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

```
template<typename Integers, size_t i>
struct my_coeff_at {
    using type = typename FractionField<Integers>::one;
};

template<typename Integers, size_t degree>
    using my_serie = taylor<Integers, my_coeff_at, degree>;

static constexpr double x = my_serie<i64, 3>::eval(3.0);
```

On x86-64 and CUDA platforms at least, using proper compiler directives, these functions yield very performant assembly, similar or better than standard library implementation in fast math. For example, this code:

```
double compute_expm1(const size_t N, double* in, double* out) {
   using V = aerobus::expm1<aerobus::i64, 13>;
   for (size_t i = 0; i < N; ++i) {
      out[i] = V::eval(in[i]);
   }
}</pre>
```

Yields this assembly (clang 17, -mavx2 -03) where we can see a pile of Fused-Multiply-Add vector instructions, generated because we unrolled completely the Horner evaluation loop:

```
ompute_expm1(unsigned long, double const*, double*):
          rax, [rdi-1]
  cmp
          rax, 2
  ibe
          .L5
 mov
          rcx, rdi
          eax, eax
  vxorpd xmm1, xmm1, xmm1
 vbroadcastsd ymm14, QWORD PTR .LC1[rip]
vbroadcastsd ymm13, QWORD PTR .LC3[rip]
shr rcx, 2
 vbroadcastsd ymm12, QWORD PTR .LC5[rip] vbroadcastsd ymm11, QWORD PTR .LC7[rip]
          rcx, 5
  vbroadcastsd ymm10, QWORD PTR .LC9[rip]
 vbroadcastsd
                   ymm9, QWORD PTR .LC11[rip]
 vbroadcastsd ymm8, QWORD PTR .LC13[rip] vbroadcastsd ymm7, QWORD PTR .LC15[rip]
  vbroadcastsd
                  ymm6, QWORD PTR .LC17[rip]
 vbroadcastsd
vbroadcastsd
                   ymm5, QWORD PTR .LC19[rip]
                   ymm4, QWORD PTR .LC21[rip]
 vbroadcastsd
                  ymm3, QWORD PTR .LC23[rip]
  vbroadcastsd
                   ymm2, QWORD PTR .LC25[rip]
.L3:
  vmovupd ymm15, YMMWORD PTR [rsi+rax]
  vmovapd ymm0, ymm15
  vfmadd132pd
                   ymm0, ymm14, ymm1
 vfmadd132pd
                   ymm0, ymm13, ymm15
  vfmadd132pd
                   ymm0, ymm12, ymm15
  vfmadd132pd
                   ymm0, ymm11, ymm15
  vfmadd132pd
                   ymm0, ymm10, ymm15
  vfmadd132pd
                   ymm0, ymm9, ymm15
                   ymm0, ymm8, ymm15
  vfmadd132pd
 vfmadd132pd
                   ymm0, ymm7, ymm15
 vfmadd132pd
                   ymm0, ymm6, ymm15
  vfmadd132pd
                   ymm0, ymm5, ymm15
 vfmadd132pd
                   ymm0, ymm4, ymm15
  vfmadd132pd
                   ymm0, ymm3, ymm15
  vfmadd132pd
                   ymm0, ymm2, ymm15
 vfmadd132pd
                   ymm0, ymm1, ymm15
  vmovupd YMMWORD PTR [rdx+rax], ymm0
          rax, 32
 add
  cmp
          rcx, rax
  jne
          .L3
          rax, rdi
  and
          rax,
 vzeroupper
```

#### 1.3 Operations

#### 1.3.1 Field of fractions

Given a set (type) satisfies the IsEuclideanDomain concept, Aerobus allows to define its field of fractions.

6 Introduction

This new type is again a euclidean domain, especially a field, and therefore we can define polynomials over it.

For example, integers modulo p is not a field when p is not prime. We then can define its field of fraction and polynomials over it this way:

```
using namespace aerobus;
using ZmZ = zpz<8>;
using Fzmz = FractionField<ZmZ>;
using Pfzmz = polynomial<Fzmz>;
```

The same operation would stand for any set that users would have implemented in place of ZmZ.

For example, we can easily define rational functions by taking the ring of fractions of polynomials: using namespace aerobus; using RF64 = FractionField<polynomial<q64>>;

Which also have an evaluation function, as polynomial do.

#### 1.3.2 Quotient

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

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

As we could have using zpz<17>.

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

#### 1.4 Misc

#### 1.4.1 Continued Fractions

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

As practical examples, aerobus gives continued fractions of  $\pi$ , e,  $\sqrt{2}$  and  $\sqrt{3}$ : constexpr double A\_SQRT3 = aerobus::SQRT3\_fraction::val; // 1.7320508075688772935

#### **1.5 CUDA**

When compiled with nvcc and the flag WITH\_CUDA\_FP16, Aerobus provides some kind of support of 16 bits integers and floats (aka  $\__half$ ).

Unfortunately, NVIDIA did not put enough constexpr in its <code>cuda\_fp16.h</code> header, so we had to implement our own constexpr static\_cast from int16\_t to <code>\_\_half</code> to make integers polynomials work with <code>\_\_half</code>. See <code>thisbug</code>.

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

Please push to make these bug fixed by NVIDIA.

# **Namespace Index**

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

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

8 Namespace Index

# **Concept Index**

### 3.1 Concepts

Here is a list of all concepts with brief descriptions:

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

10 Concept Index

## **Class Index**

### 4.1 Class List

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

aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, E >	43
aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable_if_t < (index < 0  index > 0) > >	>
43	
aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable_if_t < (index==0) >>	44
aerobus::ContinuedFraction < values >	
Continued fraction a0 + $\frac{1}{a_1 + \frac{1}{a_2 + \dots}}$	44
aerobus::ContinuedFraction $<$ a0 $>$	
Specialization for only one coefficient, technically just 'a0'	45
aerobus::ContinuedFraction < a0, rest >	
Specialization for multiple coefficients (strictly more than one)	46
aerobus::ConwayPolynomial	47
aerobus::Embed < Small, Large, E >	
Embedding - struct forward declaration	47
aerobus::Embed < i32, i64 >	
Embeds i32 into i64	48
aerobus::Embed< polynomial< Small >, polynomial< Large >>	
Embeds polynomial < Small > into polynomial < Large >	49
aerobus::Embed < q32, q64 >	
Embeds q32 into q64	50
aerobus::Embed< Quotient< Ring, X >, Ring >	
Embeds Quotient < Ring, X > into Ring	50
aerobus::Embed< Ring, FractionField< Ring >>	
Embeds values from Ring to its field of fractions	51
aerobus::Embed< zpz< x >, i32 >	
Embeds zpz values into i32	52
aerobus::polynomial< Ring >::horner_reduction_t< P >	
Used to evaluate polynomials over a value in Ring	53
aerobus::i32	
32 bits signed integers, seen as a algebraic ring with related operations	54
aerobus::i64	
64 bits signed integers, seen as a algebraic ring with related operations	59
aerobus::polynomial < Ring >::horner_reduction_t < P >::inner < index, stop >	65
$aerobus::polynomial < Ring > ::horner\_reduction\_t < P > ::inner < stop, stop > \dots \dots \dots \dots \dots$	66
aerobus::is_prime< n >	
Checks if n is prime	67

12 Class Index

aerobus::polynomial< Ring >	67
aerobus::type_list< Ts >::pop_front	
Removes types from head of the list	75
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	76
aerobus::type_list< Ts >::split< index >	
Splits list at index	81
aerobus::type_list< Ts >	
Empty pure template struct to handle type list	82
aerobus::type_list<>	
Specialization for empty type list	85
aerobus::i32::val< x >	
Values in i32, again represented as types	86
aerobus::i64::val< x >	
Values in i64	88
aerobus::polynomial< Ring >::val< coeffN, coeffs >	
Values (seen as types) in polynomial ring	90
aerobus::Quotient< Ring, X >::val< V >	
Projection values in the quotient ring	94
aerobus::zpz::val< x >	
Values in zpz	94
aerobus::polynomial< Ring >::val< coeffN >	
Specialization for constants	97
aerobus::zpz	
Congruence classes of integers modulo n (32 hits)	101

## **File Index**

### 5.1 File List

Here is a list of all files with brief descriptions:

src/aerobus.h .			 								 												109
src/examples.h			 								 												200

14 File Index

## **Namespace Documentation**

#### 6.1 aerobus Namespace Reference

main namespace for all publicly exposed types or functions

#### **Namespaces**

- · namespace internal
  - internal implementations, subject to breaking changes without notice
- namespace known\_polynomials

families of well known polynomials such as Hermite or Bernstein

#### Classes

```
• struct ContinuedFraction
```

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

struct ContinuedFraction < a0 >

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

- struct ContinuedFraction< a0, rest... >
  - specialization for multiple coefficients (strictly more than one)
- · struct ConwayPolynomial
- struct Embed

```
embedding - struct forward declaration
```

struct Embed< i32, i64 >

embeds i32 into i64

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

embeds polynomial<Small> into polynomial<Large>

struct Embed< q32, q64 >

embeds q32 into q64

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

embeds Quotient<Ring, X> into Ring

struct Embed< Ring, FractionField< Ring > >

embeds values from Ring to its field of fractions

struct Embed< zpz< x >, i32 >

embeds zpz values into i32

• struct i32

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

struct i64

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

• struct is\_prime

checks if n is prime

- struct polynomial
- struct Quotient

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

struct type list

Empty pure template struct to handle type list.

struct type\_list<>

specialization for empty type list

struct zpz

congruence classes of integers modulo p (32 bits)

#### Concepts

· concept IsRing

Concept to express R is a Ring.

• concept IsEuclideanDomain

Concept to express R is an euclidean domain.

concept IsField

Concept to express R is a field.

#### **Typedefs**

```
• template<typename T , typename A , typename B >
  using gcd_t = typename internal::gcd< T >::template type< A, B >
     computes the greatest common divisor or A and B
• template<typename... vals>
  using vadd_t = typename internal::vadd< vals... >::type
     adds multiple values (v1 + v2 + ... + vn) vals must have same "enclosing_type" and "enclosing_type" must have an
     add_t binary operator
• template<typename... vals>
  using vmul t = typename internal::vmul < vals... >::type
     multiplies multiplie values (v1 + v2 + ... + vn) vals must have same "enclosing_type" and "enclosing_type" must have
     an mul_t binary operator

    template<typename val >

  using abs t = std::conditional t < val::enclosing type::template pos v < val >, val, typename val::enclosing ←
  _type::template sub_t< typename val::enclosing_type::zero, val > >
     computes absolute value of 'val' val must be a 'value' in a Ring satisfying 'lsEuclideanDomain' concept

    template<typename Ring >

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

using sub\_t = typename X::enclosing\_type::template sub\_t < X, Y >

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

    template<typename X , typename Y >

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

 using q32 = FractionField < i32 >

     32 bits rationals rationals with 32 bits numerator and denominator

    using fpq32 = FractionField< polynomial< q32 >>

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

 using q64 = FractionField < i64 >

     64 bits rationals rationals with 64 bits numerator and denominator
using pi64 = polynomial < i64 >
     polynomial with 64 bits integers coefficients
using pq64 = polynomial < q64 >
     polynomial with 64 bits rationals coefficients

    using fpq64 = FractionField< polynomial< q64 > >

     polynomial with 64 bits rational coefficients

    template<typename Ring , typename v1 , typename v2 >

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

    template<typename Ring , typename v1 , typename v2 >

  using addfractions t = typename FractionField < Ring >::template add t < v1, v2 >
     helper type: adds two fractions
• template<typename Ring , typename v1 , typename v2 >
  using mulfractions_t = typename FractionField< Ring >::template mul_t< v1, v2 >
     helper type: multiplies two fractions
• template<typename Ring , auto... xs>
  using make_int_polynomial_t = typename polynomial < Ring >::template val < typename Ring::template
  inject_constant_t< xs >... >
     make a polynomial with coefficients in Ring
• template<typename Ring, auto... xs>
  using make frac polynomial t = typename polynomial < FractionField < Ring > >::template val < typename
  FractionField < Ring >::template inject_constant_t < xs >... >
     make a polynomial with coefficients in FractionField<Ring>
• template<typename T , size_t i>
  using factorial_t = typename internal::factorial < T, i >::type
     computes factorial(i), as type
```

```
• template<typename T , size_t k, size_t n>
  using combination_t = typename internal::combination < T, k, n >::type
     computes binomial coefficient (k among n) as type
• template<typename T , size_t n>
  using bernoulli t = typename internal::bernoulli < T, n >::type
     nth bernoulli number as type in T
template<typename T, size_t n>
  using bell_t = typename internal::bell_helper< T, n >::type
     Rell numbers
• template<typename T , int k>
  using alternate_t = typename internal::alternate< T, k >::type
      (-1)^{\wedge}k as type in T
• template<typename T , int n, int k>
  using stirling_1_signed_t = typename internal::stirling_1_helper< T, n, k >::type
      Stirling number of first king (signed) - as types.
• template<typename T , int n, int k>
  using stirling_1_unsigned_t = abs_t< typename internal::stirling_1_helper< T, n, k >::type >
      Stirling number of first king (unsigned) - as types.
• template<typename T , int n, int k>
  using stirling 2 t = typename internal::stirling 2 helper< T, n, k >::type
      Stirling number of second king – as types.
• template<typename T , typename p , size_t n>
  using pow_t = typename internal::pow< T, p, n >::type
     p^{\wedge}n (as 'val' type in T)

    template<typename T, template< typename, size t index > typename coeff at, size t deg>

  using taylor = typename internal::make taylor impl< T, coeff at, internal::make index sequence reverse<
  deg+1 > > :: type
• template<typename Integers , size_t deg>
  using exp = taylor < Integers, internal::exp coeff, deg >
     e^x
• template<typename Integers , size t deg>
  using expm1 = typename polynomial < FractionField < Integers > >::template sub t < exp < Integers, deg
  >, typename polynomial < FractionField < Integers > >::one >
• template<typename Integers , size_t deg>
  using lnp1 = taylor < Integers, internal::lnp1 coeff, deg >
     ln(1+x)
• template<typename Integers , size_t deg>
  using atan = taylor < Integers, internal::atan_coeff, deg >
     \arctan(x)
• template<typename Integers , size_t deg>
  using sin = taylor < Integers, internal::sin coeff, deg >
     \sin(x)
• template<typename Integers , size_t deg>
  using sinh = taylor < Integers, internal::sh_coeff, deg >
• template<typename Integers , size_t deg>
  using cosh = taylor < Integers, internal::cosh coeff, deg >
     \cosh(x) hyperbolic cosine
• template<typename Integers , size_t deg>
  using cos = taylor < Integers, internal::cos coeff, deg >
     cos(x) cosinus

    template<typename Integers , size_t deg>

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

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

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

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

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

#### **Functions**

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

#### **Variables**

#### 6.1.1 Detailed Description

main namespace for all publicly exposed types or functions

#### 6.1.2 Typedef Documentation

#### 6.1.2.1 abs t

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

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

#### **Template Parameters**

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

#### 6.1.2.2 add\_t

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

#### generic addition

#### **Template Parameters**

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

#### 6.1.2.3 addfractions\_t

```
template<typename Ring , typename v1 , typename v2 >
using aerobus::addfractions_t = typedef typename FractionField<Ring>::template add_t<v1, v2>
```

#### helper type: adds two fractions

#### **Template Parameters**

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

#### 6.1.2.4 alternate\_t

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

#### **Template Parameters**

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

#### 6.1.2.5 asin

```
template<typename Integers , size_t deg> using aerobus::asin = typedef taylor<Integers, internal::asin_coeff, deg> \arcsin(x) arc sinus
```

#### **Template Parameters**

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

#### 6.1.2.6 asinh

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

#### **Template Parameters**

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

#### 6.1.2.7 atan

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

#### **Template Parameters**

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

#### 6.1.2.8 atanh

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

 $\operatorname{arctanh}(x)$  arc hyperbolic tangent

#### **Template Parameters**

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

#### 6.1.2.9 bell\_t

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

Bell numbers.

#### **Template Parameters**

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

#### 6.1.2.10 bernoulli\_t

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

nth bernoulli number as type in T

#### **Template Parameters**

T	Ring type (i64)
n	

#### 6.1.2.11 combination\_t

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

computes binomial coefficient (k among n) as type

#### **Template Parameters**

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

#### 6.1.2.12 cos

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

 $\cos(x)$  cosinus

# **Template Parameters**

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

#### 6.1.2.13 cosh

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

#### **Template Parameters**

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

# 6.1.2.14 div\_t

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

#### generic division

#### **Template Parameters**

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

### 6.1.2.15 E\_fraction

```
using aerobus::E_fraction = typedef ContinuedFraction<2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1 > 0
```

approximation of  $\boldsymbol{e}$ 

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

embed a polynomial with integers coefficients into rational coefficients polynomials

Lives in polynomial<FractionField<Ring>>

# **Template Parameters**

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

#### 6.1.2.17 exp

```
template<typename Integers , size_t deg> using aerobus::exp = typedef taylor<Integers, internal::exp_coeff, deg> e^x
```

# **Template Parameters**

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

# 6.1.2.18 expm1

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

# **Template Parameters**

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

# 6.1.2.19 factorial\_t

```
template<typename T , size_t i>
using aerobus::factorial_t = typedef typename internal::factorial<T, i>::type
```

# computes factorial(i), as type

# **Template Parameters**

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<typename T , typename A , typename B >
using aerobus::gcd_t = typedef typename internal::gcd<T>::template type<A, B>
```

computes the greatest common divisor or A and B

# **Template Parameters**

```
T Ring type (must be euclidean domain)
```

# 6.1.2.24 geometric\_sum

```
template<typename Integers , size_t deg> using aerobus::geometric_sum = typedef taylor<Integers, internal::geom_coeff, deg> \frac{1}{1-x} \text{ zero development of } \frac{1}{1-x}
```

#### **Template Parameters**

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

#### 6.1.2.25 Inp1

```
template<typename Integers , size_t deg> using aerobus::lnp1 = typedef taylor<Integers, internal::lnp1_coeff, deg> \ln(1+x)
```

# **Template Parameters**

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

# 6.1.2.26 make\_frac\_polynomial\_t

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

make a polynomial with coefficients in FractionField<Ring>

#### **Template Parameters**

Ring	integers
xs	values

# 6.1.2.27 make\_int\_polynomial\_t

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

make a polynomial with coefficients in Ring

#### **Template Parameters**

Ring	integers
xs	coefficients

# 6.1.2.28 make\_q32\_t

```
template<int32_t p, int32_t q>
using aerobus::make_q32_t = typedef typename q32::template simplify_t< typename q32::val<i32::inject_constant
i32::inject_constant_t<q> >>
```

helper type: make a fraction from numerator and denominator

# **Template Parameters**

р	numerator
q	denominator

# 6.1.2.29 make\_q64\_t

```
template<int64_t p, int64_t q>
using aerobus::make_q64_t = typedef typename q64::template simplify_t< typename q64::val<i64::inject_constant
i64::inject_constant_t<q> >>
```

helper type: make a fraction from numerator and denominator

# **Template Parameters**

р	numerator
q	denominator

# 6.1.2.30 makefraction\_t

```
template<typename Ring , typename v1 , typename v2 >
using aerobus::makefraction_t = typedef typename FractionField<Ring>::template val<v1, v2>
```

helper type: the rational V1/V2 in the field of fractions of Ring

# **Template Parameters**

Ring	the base ring
v1	value 1 in Ring
v2	value 2 in Ring

# 6.1.2.31 mul\_t

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

# generic multiplication

# **Template Parameters**

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

# 6.1.2.32 mulfractions\_t

```
template<typename Ring , typename v1 , typename v2 >
using aerobus::mulfractions_t = typedef typename FractionField<Ring>::template mul_t<v1, v2>
```

helper type: multiplies two fractions

# **Template Parameters**

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

#### 6.1.2.33 pi64

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

polynomial with 64 bits integers coefficients

# 6.1.2.34 PI\_fraction

```
using aerobus::PI_fraction = typedef ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1>
```

representation of  $\pi$  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

approximation of  $\sqrt{2}$ 

# 6.1.2.42 SQRT3\_fraction

```
using aerobus::SQRT3_fraction = typedef ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2
```

approximation of

# 6.1.2.43 stirling\_1\_signed\_t

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

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

# **Template Parameters**

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

# 6.1.2.44 stirling\_1\_unsigned\_t

```
template<typename T , int n, int k>
using aerobus::stirling_1_unsigned_t = typedef abs_t<typename internal::stirling_1_helper<T,
n, k>::type>
```

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

# **Template Parameters**

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

# 6.1.2.45 stirling\_2\_t

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

Stirling number of second king – as types.

# **Template Parameters**

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

#### 6.1.2.46 sub\_t

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

# generic subtraction

# **Template Parameters**

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

#### 6.1.2.47 tan

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

# **Template Parameters**

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

#### 6.1.2.48 tanh

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

#### tanh(x) hyperbolic tangent

### **Template Parameters**

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

# 6.1.2.49 taylor

```
template<typename T , template< typename, size_t index > typename coeff_at, size_t deg>
using aerobus::taylor = typedef typename internal::make_taylor_impl< T, coeff_at, internal::make_index_sequen
+ 1> >::type
```

# **Template Parameters**

T	Used Ring type (aerobus::i64 for example)
coeff⇔	- implementation giving the 'value' (seen as type in FractionField <t></t>
_at	
deg	

# Generated by Doxygen

# 6.1.2.50 vadd\_t

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

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

# **Template Parameters**

```
...vals
```

# 6.1.2.51 vmul\_t

```
template<typename... vals>
using aerobus::vmul_t = typedef typename internal::vmul<vals...>::type
```

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

# **Template Parameters**



# 6.1.3 Function Documentation

# 6.1.3.1 aligned\_malloc()

'portable' aligned allocation of count elements of type T

# **Template Parameters**

T the type of elements to store

# **Parameters**

count	the number of elements
alignment	boundary

#### 6.1.3.2 field()

brief Conway polynomials tparam p characteristic of the aerobus::field (

prime number )

# 6.1.4 Variable Documentation

#### 6.1.4.1 alternate v

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

# (-1)<sup>∧</sup>k as value from T

#### **Template Parameters**

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

# 6.1.4.2 bernoulli\_v

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

nth bernoulli number as value in FloatType

# **Template Parameters**

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

# 6.1.4.3 combination\_v

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

computes binomial coefficients (k among n) as value

#### **Template Parameters**

T	(aerobus::i32 for example)
k	
n	

#### 6.1.4.4 factorial\_v

```
template<typename T , size_t i>
constexpr T::inner_type aerobus::factorial_v = internal::factorial<T, i>::value [inline],
[constexpr]
```

computes factorial(i) as value in T

#### **Template Parameters**

T	(aerobus::i64 for example)
i	

# 6.2 aerobus::internal Namespace Reference

internal implementations, subject to breaking changes without notice

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

#### **Classes**

```
    struct FractionField

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

• struct _is_prime
struct _is_prime< 0, i >

    struct _is_prime< 1, i >

• struct _{\mbox{is\_prime}}< 2, i >

    struct _is_prime< 3, i >

    struct _is_prime< 5, i >

• struct _{\bf is\_prime}< 7, i >

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

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

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

    struct AbelHelper

• struct AllOneHelper

    struct AllOneHelper< 0, I >

· struct alternate

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

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

· struct asin_coeff

    struct asin_coeff_helper

struct asin_coeff_helper< T, i, std::enable_if_t<(i &1)==0 >>
struct asin_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >>
· struct asinh coeff
· struct asinh_coeff_helper
struct asinh_coeff_helper< T, i, std::enable_if_t<(i &1)==0 >>
```

 struct atan\_coeff struct atan\_coeff\_helper struct atan\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==0 >> struct atan\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==1 >> · struct atanh coeff struct atanh\_coeff\_helper struct atanh coeff helper< T, i, std::enable if t<(i &1)==0>> struct atanh\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==1 >> struct bell\_helper • struct bell\_helper< T, 0 > struct bell\_helper< T, 1 > struct bell\_helper< T, n, std::enable\_if\_t<(n > 1)>> struct bernoulli struct bernoulli < T, 0 > struct bernoulli\_coeff • struct bernoulli helper struct bernoulli\_helper< T, accum, m, m > struct bernstein helper struct bernstein\_helper< 0, 0, I > • struct bernstein\_helper< i, m, l, std::enable\_if\_t<(m > 0) &&(i > 0) &&(i < m)> > struct bernstein\_helper< i, m, I, std::enable\_if\_t<(m > 0) &&(i==0)> > struct bernstein\_helper< i, m, I, std::enable\_if\_t<(m > 0) &&(i==m)> > • struct BesselHelper struct BesselHelper< 0, I > struct BesselHelper< 1, I > struct chebyshev\_helper struct chebyshev\_helper< 1, 0, I > struct chebyshev\_helper< 1, 1, I > struct chebyshev helper< 2, 0, I > struct chebyshev\_helper< 2, 1, I > • struct combination struct combination helper struct combination\_helper< T, 0, n > • struct combination helper < T, k, n, std::enable if t<(n >=0 &&k >(n/2) &&k > 0)>struct combination\_helper< T, k, n, std::enable\_if\_t<(n >=0 &&k<=(n/2) &&k > 0)> > struct cos\_coeff struct cos coeff helper struct cos\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==0 >> struct cos\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==1 >> struct cosh\_coeff · struct cosh coeff helper struct cosh\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==0 >> struct cosh\_coeff\_helper< T, i, std::enable\_if\_t<(i &1)==1 >> struct exp\_coeff · struct factorial struct factorial < T, 0 > struct factorial< T, x, std::enable\_if\_t<(x > 0)>> struct fma helper struct fma\_helper< double > struct fma\_helper< float > struct fma\_helper< int16\_t > struct fma\_helper< int32\_t > struct fma helper< int64 t > struct FractionFieldImpl struct FractionFieldImpl< Field, std::enable\_if\_t< Field::is\_field >>

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

    struct gcd

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

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

    struct geom_coeff

    struct hermite helper

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

    struct hermite helper< 0, known polynomials::hermite kind::probabilist, I >

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

    struct hermite helper< 1, known polynomials::hermite kind::probabilist, I >

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

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

    struct insert h

· struct is instantiation of

 struct is instantiation of< TT, TT< Ts... >>

    struct laguerre_helper

    struct laguerre_helper< 0, I >

    struct laguerre_helper< 1, I >

· struct legendre helper
• struct legendre helper < 0, I >

    struct legendre_helper< 1, I >

    struct Inp1_coeff

struct Inp1_coeff< T, 0 >
• struct make_taylor_impl
• struct make_taylor_impl< T, coeff_at, std::integer_sequence< size_t, ls... >>
struct pop_front_h
· struct pow
struct pow< T, p, n, std::enable_if_t< n==0 >>

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

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

    struct pow scalar

    struct remove h

· struct sh coeff
• struct sh_coeff_helper

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

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

    struct sin_coeff

    struct sin coeff helper

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

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

· struct split h

    struct split_h< 0, L1, L2 >

· struct 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

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

# 7.1.2 Detailed Description

Concept to express R is an euclidean domain.

# 7.2 aerobus::IsField Concept Reference

Concept to express R is a field.

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

# 7.2.1 Concept definition

# 7.2.2 Detailed Description

Concept to express R is a field.

# 7.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring.

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

# 7.3.1 Concept definition

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

# 7.3.2 Detailed Description

Concept to express R is a Ring.

# **Chapter 8**

# **Class Documentation**

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

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

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

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

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

#### **Public Types**

• using type = typename Ring::zero

# 8.2.1 Member Typedef Documentation

# 8.2.1.1 type

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

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

• src/aerobus.h

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

#include <aerobus.h>

# **Public Types**

using type = aN

# 8.3.1 Member Typedef Documentation

#### 8.3.1.1 type

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

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

• src/aerobus.h

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

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

#include <aerobus.h>

# 8.4.1 Detailed Description

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

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

#### **Template Parameters**

values	are
	int64_t

# **Examples**

examples/continued\_fractions.cpp.

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

• src/aerobus.h

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

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

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

# **Public Types**

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

#### **Static Public Attributes**

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

# 8.5.1 Detailed Description

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

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

# **Template Parameters**

a0	an integer
	int64_t

# 8.5.2 Member Typedef Documentation

# 8.5.2.1 type

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

represented value as aerobus::q64

# 8.5.3 Member Data Documentation

#### 8.5.3.1 val

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

represented value as double

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

• src/aerobus.h

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

specialization for multiple coefficients (strictly more than one)

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

# **Public Types**

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

# **Static Public Attributes**

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

# 8.6.1 Detailed Description

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

specialization for multiple coefficients (strictly more than one)

# **Template Parameters**

a0	integer (int64_t)
rest	integers
	(int64_t)

# 8.6.2 Member Typedef Documentation

#### 8.6.2.1 type

```
template<int64_t a0, int64_t... rest>
using aerobus::ContinuedFraction< a0, rest... >::type = q64::template add_t< typename q64
::template inject_constant_t<a0>, typename q64::template div_t< typename q64::one, typename
ContinuedFraction<rest...>::type > >
```

represented value as aerobus::q64

# 8.6.3 Member Data Documentation

#### 8.6.3.1 val

```
template<int64_t a0, int64_t... rest>
constexpr double aerobus::ContinuedFraction< a0, rest... >::val = type::template get<double>()
[static], [constexpr]
```

reprensented value as double

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

• src/aerobus.h

# 8.7 aerobus::ConwayPolynomial Struct Reference

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

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

src/aerobus.h

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

embedding - struct forward declaration

# 8.8.1 Detailed Description

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

embedding - struct forward declaration

# **Template Parameters**

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

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

• src/aerobus.h

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

```
embeds i32 into i64
```

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

# **Public Types**

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

# 8.9.1 Detailed Description

embeds i32 into i64

# 8.9.2 Member Typedef Documentation

# 8.9.2.1 type

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

the i64 representation of val

#### **Template Parameters**

```
val a value in i32
```

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

· src/aerobus.h

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

embeds polynomial<Small> into polynomial<Large>

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

# **Public Types**

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

the polynomial<Large> reprensentation of v

# 8.10.1 Detailed Description

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

embeds polynomial<Small> into polynomial<Large>

#### **Template Parameters**

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

# 8.10.2 Member Typedef Documentation

# 8.10.2.1 type

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

the polynomial<Large> reprensentation of v

#### **Template Parameters**

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

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

· src/aerobus.h

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

embeds q32 into q64

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

# **Public Types**

```
    template<typename v >
        using type = make_q64_t< static_cast< int64_t >(v::x::v), static_cast< int64_t >(v::y::v)>
        q64 representation of v
```

# 8.11.1 Detailed Description

embeds q32 into q64

# 8.11.2 Member Typedef Documentation

# 8.11.2.1 type

q64 representation of v

**Template Parameters** 

```
v a value in q32
```

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

• src/aerobus.h

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

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

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

#### **Public Types**

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

# 8.12.1 Detailed Description

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

embeds Quotient<Ring, X> into Ring

#### **Template Parameters**

Ring	a Euclidean ring
X	a value in Ring

# 8.12.2 Member Typedef Documentation

# 8.12.2.1 type

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

Ring reprensentation of val.

# **Template Parameters**

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

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

• src/aerobus.h

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

embeds values from Ring to its field of fractions

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

# **Public Types**

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

# 8.13.1 Detailed Description

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

embeds values from Ring to its field of fractions

# **Template Parameters**

Ring an integers ring, such as i32

# 8.13.2 Member Typedef Documentation

#### 8.13.2.1 type

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

FractionField<Ring> reprensentation of v.

#### **Template Parameters**

```
v a Ring value
```

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

• src/aerobus.h

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

embeds zpz values into i32

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

# **Public Types**

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

# 8.14.1 Detailed Description

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

embeds zpz values into i32

# **Template Parameters**

x an integer

# 8.14.2 Member Typedef Documentation

#### 8.14.2.1 type

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

the i32 reprensentation of val

#### **Template Parameters**

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

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

• src/aerobus.h

# 8.15 aerobus::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 >

# 8.15.1 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

# 8.16 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

# 8.16.1 Detailed Description

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

# 8.16.2 Member Typedef Documentation

# 8.16.2.1 add t

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

addition operator yields v1 + v2

# **Template Parameters**

v1	a value in i32
v2	a value in i32

# 8.16.2.2 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

# 8.16.2.3 eq t

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

equality operator (type) yields v1 == v2 as std::integral\_constant<bool>

# **Template Parameters**

v1	a value in i32
v2	a value in i32

# 8.16.2.4 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 i32
v2	a value in i32

# 8.16.2.5 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

# 8.16.2.6 inject\_constant\_t

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

inject a native constant

# **Template Parameters**



# 8.16.2.7 inject\_ring\_t

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

# 8.16.2.8 inner\_type

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

# 8.16.2.9 lt\_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

# 8.16.2.10 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

# 8.16.2.11 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

#### 8.16.2.12 one

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

constant one

# 8.16.2.13 pos\_t

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

positivity operator yields v > 0 as std::true\_type or std::false\_type

**Template Parameters** 

```
v a value in i32
```

# 8.16.2.14 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

# 8.16.2.15 zero

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

constant zero

# 8.16.3 Member Data Documentation

# 8.16.3.1 eq\_v

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

equality operator (boolean value)

# **Template Parameters**

v1	
v2	

# 8.16.3.2 is\_euclidean\_domain

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

integers are an euclidean domain

#### 8.16.3.3 is\_field

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

integers are not a field

# 8.16.3.4 pos\_v

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

positivity (boolean value) yields v > 0 as boolean value

#### **Template Parameters**

```
v a value in i32
```

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

• src/aerobus.h

# 8.17 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 It v = It t < v1, v2 > ::value
      strictly smaller operator yields v1 < v2 as boolean value

    template<typename v1 , typename v2 >

  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
```

## 8.17.1 Detailed Description

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

## 8.17.2 Member Typedef Documentation

## 8.17.2.1 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

#### 8.17.2.2 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

#### 8.17.2.3 eq\_t

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

equality operator yields v1 == v2 as std::true\_type or std::false\_type

#### **Template Parameters**

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

### 8.17.2.4 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

#### 8.17.2.5 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

#### 8.17.2.6 inject\_constant\_t

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

injects constant as an i64 value

#### **Template Parameters**



# 8.17.2.7 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
```

## 8.17.2.8 inner\_type

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

type of represented values

#### 8.17.2.9 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

## 8.17.2.10 mod\_t

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

modulus operator

#### **Template Parameters**

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

#### 8.17.2.11 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

#### 8.17.2.12 one

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

constant one

# 8.17.2.13 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
```

## 8.17.2.14 sub\_t

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

#### substraction operator

#### **Template Parameters**

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

#### 8.17.2.15 zero

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

constant zero

## 8.17.3 Member Data Documentation

## 8.17.3.1 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

## 8.17.3.2 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

#### 8.17.3.3 is\_euclidean\_domain

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

integers are an euclidean domain

#### 8.17.3.4 is\_field

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

integers are not a field

#### 8.17.3.5 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

#### **Template Parameters**

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

#### 8.17.3.6 pos v

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

positivity yields v > 0 as boolean value

#### **Template Parameters**

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

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

• src/aerobus.h

# 8.18 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 >
```

## 8.18.1 Member Typedef Documentation

#### 8.18.1.1 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_\(\cup \)
at_t<P::degree - index> >, x>
```

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

• src/aerobus.h

# 8.19 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
```

## 8.19.1 Member Typedef Documentation

#### 8.19.1.1 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

# 8.20 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

## 8.20.1 Detailed Description

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

Template Parameters
```

#### 8.20.2 Member Data Documentation

## 8.20.2.1 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

# 8.21 aerobus::polynomial < Ring > Struct Template Reference

#include <aerobus.h>

#### Classes

· struct horner\_reduction\_t

Used to evaluate polynomials over a value in Ring.

struct val

values (seen as types) in polynomial ring

struct val< coeffN >

specialization for constants

#### **Public Types**

```
    using zero = val< typename Ring::zero >

     constant zero
using one = val< typename Ring::one >
     constant one

    using X = val< typename Ring::one, typename Ring::zero >

     generator

    template<typename P >

  using simplify t = typename simplify < P >::type
     simplifies a polynomial (recursively deletes highest degree if zero, do nothing otherwise)
• template<typename v1 , typename v2 >
  using add_t = typename add< v1, v2 >::type
     adds two polynomials

    template<typename v1 , typename v2 >

  using sub_t = typename sub< v1, v2 >::type
     substraction of two polynomials
• template<typename v1 , typename v2 >
  using mul_t = typename mul < v1, v2 >::type
     multiplication of two polynomials
• template<typename v1 , typename v2 >
  using eq_t = typename eq_helper< v1, v2 >::type
     equality operator
• template<typename v1 , typename v2 >
  using It_t = typename It_helper< v1, v2 >::type
     strict less operator
• template<typename v1, typename v2 >
  using gt_t = typename gt_helper< v1, v2 >::type
     strict greater operator
• template<typename v1 , typename v2 >
  using div_t = typename div < v1, v2 >::q_type
     division operator

    template<typename v1 , typename v2 >

  using mod_t = typename div_helper< v1, v2, zero, v1 >::mod_type
     modulo operator
• template<typename coeff , size_t deg>
  using monomial_t = typename monomial < coeff, deg >::type
     monomial : coeff X^{\wedge} deg

    template<typename v >

  using derive_t = typename derive_helper< v >::type
     derivation operator

    template<typename v >

  using pos_t = typename Ring::template pos_t < typename v::aN >
     checks for positivity (an > 0)

    template<typename v1 , typename v2 >

  using gcd t = std::conditional t < Ring::is euclidean domain, typename make unit < gcd t < polynomial <
  Ring >, v1, v2 > ::type, void >
     greatest common divisor of two polynomials

    template<auto x>

  using inject_constant_t = val< typename Ring::template inject_constant_t < x > >
     makes the constant (native type) polynomial a_0

    template<typename v >

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

#### **Static Public Attributes**

```
• static constexpr bool is_field = false
```

- static constexpr bool is\_euclidean\_domain = Ring::is\_euclidean\_domain
- template<typename v >
   static constexpr bool pos\_v = pos\_t<v>::value
   positivity operator

## 8.21.1 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/make\_polynomial.cpp, and examples/modular\_arithmetic.cpp.

## 8.21.2 Member Typedef Documentation

#### 8.21.2.1 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	

## 8.21.2.2 derive\_t

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

#### derivation operator

#### **Template Parameters**



#### 8.21.2.3 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	

#### 8.21.2.4 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	

#### 8.21.2.5 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	

## 8.21.2.6 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	

#### 8.21.2.7 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**

X	

#### 8.21.2.8 inject\_ring\_t

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

makes the constant (ring type) polynomial a\_0

#### **Template Parameters**



## 8.21.2.9 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	

#### 8.21.2.10 mod t

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

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

#### modulo operator

#### **Template Parameters**

v1	
v2	

## 8.21.2.11 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	

## 8.21.2.12 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	

#### 8.21.2.13 one

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

#### constant one

## 8.21.2.14 pos\_t

```
template<typename Ring >
template<typename v >
using aerobus::polynomial< Ring >::pos_t = typename Ring::template pos_t<typename v::aN>
```

checks for positivity (an > 0)

#### **Template Parameters**

V	-
---	---

#### 8.21.2.15 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**



#### 8.21.2.16 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	

#### 8.21.2.17 X

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

#### generator

#### 8.21.2.18 zero

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

## constant zero

#### 8.21.3 Member Data Documentation

#### 8.21.3.1 is euclidean domain

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

#### 8.21.3.2 is field

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

## 8.21.3.3 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

# 8.22 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

## 8.22.1 Detailed Description

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

removes types from head of the list

## 8.22.2 Member Typedef Documentation

#### 8.22.2.1 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

#### 8.22.2.2 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

# 8.23 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 > zaddition operator
    template<typename v1 , typename v2 > using mul_t = val< typename Ring::template mul_t< typename v1::type, typename v2::type > substraction operator
    template<typename v1 , typename v2 > using div_t = val< typename Ring::template div_t< typename v1::type, typename v2::type > division operator
    template<typename v1 , typename Ring::template div_t< typename v1::type, typename v2::type > division operator
    template<typename v1 , typename v2 > template
```

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

## 8.23.1 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**

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

## 8.23.2 Member Typedef Documentation

#### 8.23.2.1 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 \leftrightarrow ::type, typename v2::type> >
```

#### addition operator

## **Template Parameters**

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

#### 8.23.2.2 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 \leftrightarrow ::type, typename v2::type> >
```

#### division operator

#### **Template Parameters**

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

#### 8.23.2.3 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

## 8.23.2.4 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

#### 8.23.2.5 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
```

#### 8.23.2.6 mod\_t

```
template<typename Ring , typename X > template<typename v1 , typename v2 > using aerobus::Quotient< Ring, X >::mod_t = val<typename Ring::template mod_t<typename v1 \leftarrow ::type, typename v2::type> >
```

#### modulus operator

#### **Template Parameters**

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

## 8.23.2.7 mul\_t

```
template<typename Ring , typename X > template<typename v1 , typename v2 > using aerobus::Quotient< Ring, X >::mul_t = val<typename Ring::template mul_t<typename v1 \leftarrow ::type, typename v2::type> >
```

#### substraction operator

#### **Template Parameters**

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

## 8.23.2.8 one

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

one

#### 8.23.2.9 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
```

#### 8.23.2.10 zero

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

zero value

#### 8.23.3 Member Data Documentation

#### 8.23.3.1 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

## 8.23.3.2 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
```

## 8.23.3.3 pos\_v

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

**Template Parameters** 

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

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

· src/aerobus.h

# 8.24 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

## 8.24.1 Detailed Description

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

## 8.24.2 Member Typedef Documentation

#### 8.24.2.1 head

index

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

#### 8.24.2.2 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

# 8.25 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
```

## 8.25.1 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

# 8.25.2 Member Typedef Documentation

#### 8.25.2.1 at

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

#### returns type at index

#### **Template Parameters**



#### 8.25.2.2 concat

```
template<typename... Ts>
template<typename U >
using aerobus::type_list< Ts >::concat = typename concat_h<U>::type
```

#### concatenates two list into one

## **Template Parameters**



#### 8.25.2.3 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	

#### 8.25.2.4 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** 



# 8.25.2.5 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** 



## 8.25.2.6 remove

```
template<typename... Ts>
template<size_t index>
using aerobus::type_list< Ts >::remove = typename internal::remove_h<index, type_list<Ts...>
>::type
```

removes type at index

**Template Parameters** 



## 8.25.3 Member Data Documentation

## 8.25.3.1 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

# 8.26 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

## 8.26.1 Detailed Description

specialization for empty type list

#### 8.26.2 Member Typedef Documentation

#### 8.26.2.1 concat

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

#### 8.26.2.2 insert

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

#### 8.26.2.3 push\_back

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

## 8.26.2.4 push\_front

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

## 8.26.3 Member Data Documentation

#### 8.26.3.1 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

# 8.27 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

## 8.27.1 Detailed Description

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

values in i32, again represented as types

**Template Parameters** 

```
x an actual integer
```

## 8.27.2 Member Typedef Documentation

#### 8.27.2.1 enclosing type

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

Enclosing ring type.

#### 8.27.2.2 is\_zero\_t

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

is value zero

#### 8.27.3 Member Function Documentation

## 8.27.3.1 get()

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

cast x into valueType

**Template Parameters** 

```
valueType double for example
```

#### 8.27.3.2 to\_string()

string representation of value

## 8.27.4 Member Data Documentation

#### 8.27.4.1 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

# 8.28 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

## 8.28.1 Detailed Description

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

x an actual integer

**Template Parameters** 

## 8.28.2 Member Typedef Documentation

#### 8.28.2.1 enclosing\_type

```
template<iint64_t x>
using aerobus::i64::val< x >::enclosing_type = i64
enclosing ring type
```

## 8.28.2.2 inner\_type

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

type of represented values

#### 8.28.2.3 is\_zero\_t

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

is value zero

## 8.28.3 Member Function Documentation

## 8.28.3.1 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)
```

#### 8.28.3.2 to\_string()

```
\label{template} $$ \template< int64_t x> $$ \template< int64_t x> $$ \template< x>::to_string () [inline], [static] $$
```

string representation

#### 8.28.4 Member Data Documentation

#### 8.28.4.1 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

# 8.29 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
```

#### **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

## 8.29.1 Detailed Description

```
template<typename Ring>
template<typename coeffN, typename... coeffs>
struct aerobus::polynomial< Ring>::val< coeffN, coeffs>
```

values (seen as types) in polynomial ring

#### **Template Parameters**

coeffN	high degree coefficient
coeffs	lower degree coefficients

## 8.29.2 Member Typedef Documentation

#### 8.29.2.1 aN

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

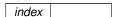
heavy weight coefficient (non zero)

#### 8.29.2.2 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**



#### 8.29.2.3 enclosing\_type

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

#### 8.29.2.4 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

## 8.29.2.5 ring\_type

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

ring coefficients live in

#### 8.29.2.6 strip

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

remove largest coefficient

## 8.29.2.7 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>
```

#### 8.29.3 Member Function Documentation

## 8.29.3.1 eval()

evaluates polynomial seen as a function operating on arithmeticType

#### **Template Parameters**

arithmeticType usually float or double
--

#### **Parameters**

```
x value
```

#### Returns

P(x)

#### 8.29.3.2 to\_string()

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

get a string representation of polynomial

#### Returns

```
something like a_n X^n + ... + a_1 X + a_0
```

#### 8.29.4 Member Data Documentation

## 8.29.4.1 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

## 8.29.4.2 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

# 8.30 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 >>

## 8.30.1 Detailed Description

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

projection values in the quotient ring

**Template Parameters** 

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

## 8.30.2 Member Typedef Documentation

#### 8.30.2.1 raw\_t

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

### 8.30.2.2 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

# 8.31 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 value Type >
    static constexpr INLINED DEVICE value Type get ()
        get value as value Type
    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
```

## 8.31.1 Detailed Description

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

Template Parameters

x an integer
```

## 8.31.2 Member Typedef Documentation

#### 8.31.2.1 enclosing\_type

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

enclosing ring type

#### 8.31.2.2 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
```

# 8.31.3 Member Function Documentation

#### 8.31.3.1 get()

```
template<int32_t p>
template<int32_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
```

## 8.31.3.2 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

## 8.31.4 Member Data Documentation

## 8.31.4.1 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

#### 8.31.4.2 v

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

actual value

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

· src/aerobus.h

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

specialization for constants

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

#### Classes

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

#### **Public Types**

```
    using ring_type = Ring
        ring coefficients live in
    using enclosing_type = polynomial < Ring >
        enclosing ring type
    using aN = coeffN
    using strip = val < coeffN >
        using is_zero_t = std::bool_constant < aN::is_zero_t::value >
        template < size_t index >
        using coeff_at_t = typename coeff_at < index > ::type
    template < typename x >
        using value_at_t = coeffN
```

# **Static Public Member Functions**

```
• static std::string to_string ()
```

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

#### **Static Public Attributes**

```
    static constexpr size_t degree = 0
        degree
```

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

98 Class Documentation

# 8.32.1 Detailed Description

template<typename Ring>
template<typename coeffN>
struct aerobus::polynomial< Ring >::val< coeffN >

specialization for constants

#### **Template Parameters**

```
coeffN
```

# 8.32.2 Member Typedef Documentation

#### 8.32.2.1 aN

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

#### 8.32.2.2 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
```

#### 8.32.2.3 enclosing\_type

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

#### enclosing ring type

# 8.32.2.4 is\_zero\_t

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

#### 8.32.2.5 ring\_type

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

#### ring coefficients live in

#### 8.32.2.6 strip

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

100 Class Documentation

#### 8.32.2.7 value\_at\_t

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

#### 8.32.3 Member Function Documentation

#### 8.32.3.1 eval()

#### 8.32.3.2 to\_string()

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

#### 8.32.4 Member Data Documentation

#### 8.32.4.1 degree

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

degree

#### 8.32.4.2 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

#### 8.33 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 >
• 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 lt_t = typename lt< 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)
```

102 Class Documentation

#### **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 It v = It 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)
```

# 8.33.1 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.

# 8.33.2 Member Typedef Documentation

#### 8.33.2.1 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

## 8.33.2.2 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

# 8.33.2.3 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

# 8.33.2.4 gcd\_t

```
template<iint32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz::gcd_t = gcd_t<i32, v1, v2>
```

#### greatest common divisor

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

104 Class Documentation

#### 8.33.2.5 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

#### 8.33.2.6 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
```

# 8.33.2.7 inner\_type

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

underlying type for values

#### 8.33.2.8 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)

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

#### 8.33.2.9 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

#### 8.33.2.10 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

#### 8.33.2.11 one

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

one value

#### 8.33.2.12 pos t

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

#### positivity operator (type)

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

106 Class Documentation

#### 8.33.2.13 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

#### 8.33.2.14 zero

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

zero value

#### 8.33.3 Member Data Documentation

# 8.33.3.1 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

# 8.33.3.2 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)

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

#### 8.33.3.3 is\_euclidean\_domain

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

always true

#### 8.33.3.4 is\_field

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

true iff p is prime

#### 8.33.3.5 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

# 8.33.3.6 pos\_v

```
template<int32_t p>
template<typename v >
constexpr bool aerobus::zpz::pos_v = pos_t < v >::value [static], [constexpr]
```

positivity operator (boolean value)

#### **Template Parameters**

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

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

• src/aerobus.h

108 Class Documentation

# **Chapter 9**

# **File Documentation**

# 9.1 README.md File Reference

# 9.2 src/aerobus.h File Reference

```
#include <cstdint>
#include <cstddef>
#include <cstring>
#include <type_traits>
#include <utility>
#include <algorithm>
#include <functional>
#include <string>
#include <concepts>
#include <array>
Include dependency graph for aerobus.h:
```

# 9.3 aerobus.h

#### Go to the documentation of this file.

```
00001 // -*- lsst-c++ -*-
00002 #ifndef __INC_AEROBUS__ // NOLINT
00003 #define __INC_AEROBUS__
00004
00005 #include <cstdint>
00006 #include <cstddef>
00007 #include <cstring>
00008 #include <type_traits>
00009 #include <utility>
00010 #include <algorithm>
00011 #include <functional>
00012 #include <string>
00013 #include <concepts> // NOLINT
00014 #include <array>
00015 #ifdef WITH_CUDA_FP16
00016 #include <bit>
00017 #include <cuda_fp16.h>
00018 #endif
00019
00023 #ifdef _MSC_VER
00024 \#define ALIGNED(x) __declspec(align(x))
00025 #define INLINED ___forceinline
00026 #else
00027 #define ALIGNED(x) __attribute__((aligned(x)))
00028 #define INLINED __attribute__((always_inline)) inline
```

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

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

```
00215
              template<>
00216
              struct fma_helper<double> {
00217
                  static constexpr INLINED DEVICE double eval(const double x, const double y, const double
     z) {
00218
                       return x * v + z;
00219
                 }
00220
              } ;
00221
00222
              template<>
00223
              struct fma_helper<float> {
                 static constexpr INLINED DEVICE float eval(const float x, const float y, const float z) {
00224
00225
                      return x * y + z;
00226
                 }
00227
              };
00228
00229
              template<>
              struct fma_helper<int32_t> {
00230
                  static constexpr INLINED DEVICE int16_t eval(const int16_t x, const int16_t y, const
00231
     int16_t z) {
00232
                      return x * y + z;
00233
00234
              } ;
00235
              template<>
00236
00237
              struct fma_helper<int16_t> {
                  static constexpr INLINED DEVICE int32_t eval(const int32_t x, const int32_t y, const
00238
     int32_t z) {
00239
                       return x * y + z;
00240
00241
              };
00242
00243
              template<>
00244
              struct fma_helper<int64_t> {
00245
                  static constexpr INLINED DEVICE int64_t eval(const int64_t x, const int64_t y, const
     int64_t z) {
00246
                       return x * y + z;
00247
                }
00248
              };
00249
00250
              #ifdef WITH_CUDA_FP16
00251
              template<>
00252
              struct fma_helper<__half> {
                  static constexpr INLINED DEVICE __half eval(const __half x, const __half y, const __half
00253
     z) {
00254
                       #ifdef ___CUDA_ARCH__
00255
                       return __hfma(x, y, z);
00256
                      #else
00257
                      return x * y + z;
00258
                       #endif
00259
                  }
00260
              };
00261
              template<>
00262
              struct fma_helper<__half2> {
00263
__half2 z) {
                  static constexpr INLINED DEVICE __half2 eval(const __half2 x, const __half2 y, const
                       #ifdef ___CUDA_ARCH_
00265
                       <u>return</u> <u>hfma2(x, y, z);</u>
00266
                       #else
00267
                      return x * y + z;
00268
                       #endif
00269
                  }
00270
              };
00271
              #endif
00272
            // namespace internal
00273 } // namespace aerobus
00274
00275 // utilities
00276 namespace aerobus {
00277
         namespace internal {
00278
              template<template<typename...> typename TT, typename T>
00279
              struct is_instantiation_of : std::false_type { };
00280
              template<template<typename...> typename TT, typename... Ts>
struct is_instantiation_of<TT, TT<Ts...» : std::true_type { };</pre>
00281
00282
00283
00284
              template<template<typename...> typename TT, typename T>
00285
              inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00286
00287
              template <int64_t i, typename T, typename... Ts>
00288
              struct type_at {
                 static_assert(i < sizeof...(Ts) + 1, "index out of range");
00289
00290
                  using type = typename type_at<i - 1, Ts...>::type;
00291
00292
00293
              template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00294
                  using type = T;
00295
              };
```

```
00296
00297
              template <size_t i, typename... Ts>
00298
              using type_at_t = typename type_at<i, Ts...>::type;
00299
00300
00301
              template<size t n, size t i, typename E = void>
00302
              struct _is_prime {};
00303
00304
              template<size_t i>
00305
              struct _is_prime<0, i> {
00306
                  static constexpr bool value = false;
00307
00308
00309
              template<size_t i>
00310
              struct _is_prime<1, i> {
00311
                 static constexpr bool value = false;
00312
              };
00313
00314
              template<size_t i>
00315
              struct _is_prime<2, i> {
00316
                 static constexpr bool value = true;
00317
00318
00319
              template<size t i>
00320
              struct _is_prime<3, i> {
00321
                 static constexpr bool value = true;
00322
00323
00324
              template<size_t i>
00325
              struct _is_prime<5, i> {
                  static constexpr bool value = true;
00326
00327
00328
00329
               template<size_t i>
00330
              struct _{is\_prime<7}, _{i>} {
                  static constexpr bool value = true;
00331
00332
              };
00333
00334
              template<size_t n, size_t i>
00335
              struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)» {
00336
                  static constexpr bool value = false;
00337
00338
00339
              template<size_t n, size_t i>
00340
              struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)» {
00341
                  static constexpr bool value = false;
00342
00343
              template<size_t n, size_t i>
00344
              struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)» {
00345
                 static constexpr bool value = true;
00346
00347
00348
00349
              {\tt template} < {\tt size\_t n, size\_t i} >
              struct _is_prime<n, i, std::enable_if_t<(
    n % i == 0 &&</pre>
00350
00351
                  n >= 9 &&
00353
                  n % 3 != 0 &&
00354
                  n % 2 != 0 &&
00355
                  i * i > n) \gg {
00356
                  static constexpr bool value = true;
00357
              };
00358
00359
              template<size_t n, size_t i>
00360
               struct _is_prime<n, i, std::enable_if_t<(
00361
                  n % (i+2) == 0 &&
00362
                  n >= 9 &&
                  n % 3 != 0 &&
00363
                  n % 2 != 0 &&
00364
00365
                  i * i <= n) » {
00366
                  static constexpr bool value = true;
00367
              };
00368
00369
              template<size_t n, size_t i>
              struct _is_prime<n, i, std::enable_if_t<(
    n % (i+2) != 0 &&</pre>
00370
00371
00372
                       n % i != 0 &&
00373
                       n >= 9 &&
                       n % 3 != 0 &&
n % 2 != 0 &&
00374
00375
00376
                       (i * i <= n)) > {
00377
                  static constexpr bool value = _is_prime<n, i+6>::value;
00378
00379
00380
          } // namespace internal
00381
00384
          template<size t n>
```

```
00385
          struct is_prime {
00387
             static constexpr bool value = internal::_is_prime<n, 5>::value;
00388
          };
00389
00393
          template<size t n>
00394
          static constexpr bool is_prime_v = is_prime<n>::value;
00395
00396
00397
          namespace internal {
00398
              template <std::size_t... Is>
              constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
00399
00400
                  -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
00401
00402
               template <std::size_t N>
00403
               using make_index_sequence_reverse
00404
                   = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00405
00411
              template<typename Ring, typename E = void>
00412
              struct gcd;
00413
               template<typename Ring>
00414
00415
               struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
00416
                  template<typename A, typename B, typename E = void>
00417
                   struct gcd_helper {};
00418
00419
                  // B = 0, A > 0
                   template<typename A, typename B>
00420
00421
                   struct gcd_helper<A, B, std::enable_if_t<
                       ((B::is_zero_t::value) &&
00422
                           (Ring::template gt_t<A, typename Ring::zero>::value))» {
00423
00424
                       using type = A;
00425
                  };
00426
00427
                   // B = 0, A < 0
00428
                   template<typename A, typename B>
                   struct gcd_helper<A, B, std::enable_if_t<
    ((B::is_zero_t::value) &&</pre>
00429
00430
                           !(Ring::template gt_t<A, typename Ring::zero>::value))» {
00431
00432
                       using type = typename Ring::template sub_t<typename Ring::zero, A>;
00433
                  };
00434
                   // B != 0
00435
                  template<typename A, typename B>
struct gcd_helper<A, B, std::enable_if_t<</pre>
00436
00437
00438
                       (!B::is_zero_t::value)
00439
00440
                   private: // NOLINT
00441
                       // A / B
                       using k = typename Ring::template div_t<A, B>;
// A - (A/B)*B = A % B
00442
00443
00444
                       using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
00445
00446
                   public:
00447
                       using type = typename gcd_helper<B, m>::type;
00448
                   };
00449
00450
                   template<typename A, typename B>
00451
                   using type = typename gcd_helper<A, B>::type;
00452
00453
          } // namespace internal
00454
          // vadd and vmul
00455
00456
          namespace internal {
00457
             template<typename... vals>
00458
              struct vmul {};
00459
00460
              template<typename v1, typename... vals>
00461
              struct vmul<v1, vals...> {
                  using type = typename v1::enclosing_type::template mul_t<v1, typename
00462
      vmul<vals...>::type>;
00463
             };
00464
00465
              template<typename v1>
00466
              struct vmul<v1> {
00467
                  using type = v1;
00468
00469
00470
              template<typename... vals>
00471
              struct vadd {};
00472
00473
              template<typename v1, typename... vals>
00474
              struct vadd<v1, vals...> {
                 using type = typename v1::enclosing_type::template add_t<v1, typename
      vadd<vals...>::type>;
00476
             };
00477
00478
              template<tvpename v1>
```

```
00479
              struct vadd<v1> {
00480
                 using type = v1;
00481
00482
          } // namespace internal
00483
00486
          template<typename T, typename A, typename B>
          using gcd_t = typename internal::gcd<T>::template type<A, B>;
00488
00492
          {\tt template}{<}{\tt typename...}~{\tt vals}{>}
00493
          using vadd_t = typename internal::vadd<vals...>::type;
00494
          template<typename... vals>
00498
00499
          using vmul_t = typename internal::vmul<vals...>::type;
00500
00504
          template<typename val>
00505
          requires IsEuclideanDomain<typename val::enclosing_type>
00506
          using abs_t = std::conditional_t<
00507
                          val::enclosing_type::template pos_v<val>,
00508
                          val, typename val::enclosing_type::template
      sub_t<typename val::enclosing_type::zero, val>>;
00509 } // namespace aerobus
00510
00511 // embedding
00512 namespace aerobus {
00517
         template<typename Small, typename Large, typename E = void>
00518
          struct Embed;
00519 }
        // namespace aerobus
00520
00521 namespace aerobus {
00526
         template<typename Ring, typename X>
00527
          requires IsRing<Ring>
00528
          struct Quotient {
00531
             template <typename V>
00532
              struct val {
              public:
00533
                  using raw_t = V;
00534
00535
                  using type = abs_t<typename Ring::template mod_t<V, X>>;
00537
00539
              using zero = val<typename Ring::zero>;
00540
00542
              using one = val<typename Ring::one>;
00543
00547
              template<typename v1, typename v2>
00548
              using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00549
00553
              template<typename v1, typename v2>
00554
              using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00555
00559
              template<tvpename v1, tvpename v2>
00560
              using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00561
00565
              template<typename v1, typename v2>
00566
              using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00567
00571
              template<typename v1, typename v2>
using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00573
00577
              template<typename v1, typename v2>
00578
              static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00579
00583
              template<typename v1>
00584
              using pos_t = std::true_type;
00585
00589
              template<typename v>
00590
              static constexpr bool pos_v = pos_t<v>::value;
00591
00593
              static constexpr bool is euclidean domain = true;
00594
00598
              template<auto x>
00599
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00600
00604
              template < typename v >
00605
              using inject_ring_t = val<v>;
00606
          };
00607
00611
          template<typename Ring, typename X>
00612
          struct Embed<Quotient<Ring, X>, Ring> {
00615
              template<typename val>
00616
              using type = typename val::raw_t;
00617
00618 }
         // namespace aerobus
00619
00620 // type_list
00621 namespace aerobus {
00623
          template <typename... Ts>
          struct type_list;
00624
```

```
00625
           namespace internal {
00626
00627
               template <typename T, typename... Us>
00628
               struct pop_front_h {
00629
                   using tail = type_list<Us...>;
using head = T;
00630
00631
00632
00633
               template <size_t index, typename L1, typename L2>
00634
               struct split_h {
00635
                 private:
                    static_assert(index <= L2::length, "index ouf of bounds");</pre>
00636
                    using a = typename L2::pop_front::type;
using b = typename L2::pop_front::tail;
00637
00638
00639
                    using c = typename L1::template push_back<a>;
00640
00641
                public:
                    using head = typename split_h<index - 1, c, b>::head; using tail = typename split_h<index - 1, c, b>::tail;
00642
00643
00644
               };
00645
00646
               template <typename L1, typename L2>  
               struct split_h<0, L1, L2> {
00647
                    using head = L1;
00648
00649
                    using tail = L2;
00650
00651
00652
               template <size_t index, typename L, typename T>
00653
                struct insert h {
                    static_assert(index <= L::length, "index ouf of bounds");</pre>
00654
00655
                    using s = typename L::template split<index>;
00656
                    using left = typename s::head;
00657
                    using right = typename s::tail;
00658
                    using 11 = typename left::template push_back<T>;
00659
                    using type = typename ll::template concat<right>;
00660
00661
00662
               template <size_t index, typename L>
00663
               struct remove_h {
00664
                  using s = typename L::template split<index>;
                    using left = typename s::head;
using right = typename s::tail;
00665
00666
                    using rr = typename right::pop_front::tail;
using type = typename left::template concat<rr>;
00667
00668
00669
00670
           } // namespace internal
00671
00674
           template <typename... Ts>
00675
           struct type_list {
00676
           private:
00677
               template <typename T>
00678
               struct concat_h;
00679
00680
               template <typename... Us>
00681
               struct concat_h<type_list<Us...» {</pre>
00682
                   using type = type_list<Ts..., Us...>;
00683
00684
00685
            public:
00687
               static constexpr size_t length = sizeof...(Ts);
00688
00691
               template <typename T>
00692
               using push_front = type_list<T, Ts...>;
00693
00696
               template <size_t index>
00697
               using at = internal::type_at_t<index, Ts...>;
00698
00700
               struct pop_front {
00702
                    using type = typename internal::pop_front_h<Ts...>::head;
00704
                    using tail = typename internal::pop_front_h<Ts...>::tail;
00705
00706
00709
               template <typename T>
00710
               using push_back = type_list<Ts..., T>;
00711
00714
                template <typename U>
00715
               using concat = typename concat_h<U>::type;
00716
00719
               template <size_t index>
00720
                struct split {
00721
                private:
00722
                    using inner = internal::split_h<index, type_list<>, type_list<Ts...»;</pre>
00723
00724
                    using head = typename inner::head;
using tail = typename inner::tail;
00725
00726
00727
               };
```

```
00728
00732
              template <typename T, size_t index>
00733
              using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00734
00737
              template <size_t index>
              using remove = typename internal::remove_h<index, type_list<Ts...»::type;</pre>
00738
00739
         };
00740
00742
         template <>
00743
          struct type_list<> {
00744
              static constexpr size_t length = 0;
00745
00746
              template <typename T>
00747
              using push_front = type_list<T>;
00748
00749
              template <typename T>
00750
              using push_back = type_list<T>;
00751
00752
              template <typename U>
00753
              using concat = U;
00754
00755
              // TODO(jewave): assert index == 0
00756
              template <typename T, size_t index>
00757
              using insert = type_list<T>;
00758
00759 } // namespace aerobus
00760
00761 // i16
00762 #ifdef WITH_CUDA_FP16
00763 // i16
00764 namespace aerobus {
00766
         struct i16 {
00767
             using inner_type = int16_t;
              template<int16_t x>
00770
00771
              struct val {
00773
                 using enclosing_type = i16;
00775
                 static constexpr int16_t v = x;
00776
00779
                  template<typename valueType>
00780
                  static constexpr INLINED DEVICE valueType get() {
00781
                      return internal::template int16_convert_helper<valueType, x>::value();
00782
                  }
00783
00785
                  using is_zero_t = std::bool_constant<x == 0>;
00786
00788
                  static std::string to_string() {
00789
                      return std::to_string(x);
00790
                  }
00791
              };
00792
00794
              using zero = val<0>;
00796
              using one = val<1>;
00798
              static constexpr bool is_field = false;
00800
              static constexpr bool is_euclidean_domain = true;
00803
              template<auto x>
00804
              using inject constant t = val<static cast<int16 t>(x)>;
00805
00806
              template<typename v>
00807
              using inject_ring_t = v;
00808
00809
           private:
              template<typename v1, typename v2>
00810
00811
              struct add {
00812
                 using type = val<v1::v + v2::v>;
00813
              } ;
00814
00815
              template<typename v1, typename v2>
00816
              struct sub {
                  using type = val<v1::v - v2::v>;
00817
00818
00819
00820
              template<typename v1, typename v2>
00821
              struct mul {
                 using type = val<v1::v* v2::v>;
00822
00823
00824
00825
              template<typename v1, typename v2>
00826
              struct div {
00827
                  using type = val<v1::v / v2::v>;
00828
00829
00830
              template<typename v1, typename v2>
              struct remainder {
00831
00832
                  using type = val<v1::v % v2::v>;
00833
00834
00835
              template<tvpename v1, tvpename v2>
```

```
struct qt {
00837
                 using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00838
00839
00840
              template<typename v1, typename v2>
00841
              struct 1t {
                 using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00842
00843
00844
00845
              template<typename v1, typename v2>
00846
              struct eq {
                using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00847
00848
00849
00850
              template<typename v1>
00851
              struct pos {
                  using type = std::bool_constant<(v1::v > 0)>;
00852
00853
00854
00855
           public:
00860
              template<typename v1, typename v2>
00861
              using add_t = typename add<v1, v2>::type;
00862
00867
              template<typename v1, typename v2>
00868
              using sub_t = typename sub<v1, v2>::type;
00869
00874
              template<typename v1, typename v2>
00875
              using mul_t = typename mul<v1, v2>::type;
00876
00881
              template<typename v1, typename v2>
00882
              using div t = typename div<v1, v2>::type;
00883
00888
              template<typename v1, typename v2>
00889
              using mod_t = typename remainder<v1, v2>::type;
00890
00895
              template<typename v1, typename v2>
00896
              using gt_t = typename gt<v1, v2>::type;
00902
              template<typename v1, typename v2>
00903
              using lt_t = typename lt<v1, v2>::type;
00904
              template<typename v1, typename v2>
00909
00910
              using eq_t = typename eq<v1, v2>::type;
00911
00915
              template<typename v1, typename v2>
00916
              static constexpr bool eq_v = eq_t<v1, v2>::value;
00917
00922
              template<typename v1, typename v2>
              using gcd_t = gcd_t<i16, v1, v2>;
00923
00924
00928
              template<typename v>
00929
              using pos_t = typename pos<v>::type;
00930
00934
              template < typename v >
00935
              static constexpr bool pos_v = pos_t<v>::value;
00936
00937 } // namespace aerobus
00938 #endif
00939
00940 // i32
00941 namespace aerobus {
         struct i32 {
00943
00944
             using inner_type = int32_t;
00947
              template<int32_t x>
00948
              struct val {
00950
                 using enclosing_type = i32;
                  static constexpr int32_t v = x;
00952
00953
00956
                  template<tvpename valueTvpe>
                  static constexpr DEVICE valueType get() {
00957
00958
                     return static_cast<valueType>(x);
00959
00960
00962
                  using is zero t = std::bool constant<x == 0>;
00963
00965
                  static std::string to_string() {
00966
                      return std::to_string(x);
00967
00968
              };
00969
00971
              using zero = val<0>;
              using one = val<1>;
00973
00975
              static constexpr bool is_field = false;
00977
              static constexpr bool is_euclidean_domain = true;
00980
              template<auto x>
              using inject_constant_t = val<static_cast<int32_t>(x)>;
00981
00982
```

```
template<typename v>
00984
              using inject_ring_t = v;
00985
           private:
00986
              template<typename v1, typename v2>
00987
00988
              struct add {
00989
                  using type = val<v1::v + v2::v>;
00990
00991
00992
              template<typename v1, typename v2>
00993
              struct sub {
                 using type = val<v1::v - v2::v>;
00994
00995
00996
00997
              template<typename v1, typename v2>
00998
              struct mul {
                  using type = val<v1::v* v2::v>;
00999
01000
01001
01002
              template<typename v1, typename v2>
01003
              struct div {
01004
                  using type = val<v1::v / v2::v>;
01005
              }:
01006
01007
              template<typename v1, typename v2>
01008
              struct remainder {
                  using type = val<v1::v % v2::v>;
01009
01010
01011
01012
              template<typename v1, typename v2>
01013
              struct at {
01014
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01015
01016
01017
              template<typename v1, typename v2>
01018
              struct lt {
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01019
01021
01022
              template<typename v1, typename v2>
01023
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01024
01025
01026
01027
              template<typename v1>
01028
              struct pos {
01029
                  using type = std::bool_constant<(v1::v > 0)>;
01030
              };
01031
01032
           public:
              template<typename v1, typename v2>
01038
              using add_t = typename add<v1, v2>::type;
01039
01044
              template<typename v1, typename v2>
01045
              using sub_t = typename sub<v1, v2>::type;
01046
01051
              template<typename v1, typename v2>
01052
              using mul_t = typename mul<v1, v2>::type;
01053
01058
              template<typename v1, typename v2>
01059
              using div_t = typename div<v1, v2>::type;
01060
01065
              template<typename v1, typename v2>
01066
              using mod_t = typename remainder<v1, v2>::type;
01067
01072
              template<typename v1, typename v2>
01073
              using gt_t = typename gt<v1, v2>::type;
01074
01079
              template<typename v1, typename v2>
              using lt_t = typename lt<v1, v2>::type;
01080
01081
01086
              template<typename v1, typename v2>
01087
              using eq_t = typename eq<v1, v2>::type;
01088
              template<typename v1, typename v2>
static constexpr bool eq_v = eq_t<v1, v2>::value;
01092
01093
01094
01099
              template<typename v1, typename v2>
01100
              using gcd_t = gcd_t < i32, v1, v2>;
01101
              template<typename v>
01105
01106
              using pos_t = typename pos<v>::type;
01107
01111
              template<typename v>
01112
              static constexpr bool pos_v = pos_t<v>::value;
01113
01114 }
         // namespace aerobus
```

```
01115
01116 // i64
01117 namespace aerobus {
01119
        struct i64 {
             using inner type = int64 t;
01121
              template<int64_t x>
01124
01125
              struct val {
01127
                 using inner_type = int32_t;
01129
                  using enclosing_type = i64;
01131
                 static constexpr int64_t v = x;
01132
01135
                 template<typename valueType>
                  static constexpr INLINED DEVICE valueType get() {
01136
01137
                      return static_cast<valueType>(x);
01138
01139
                  using is zero t = std::bool constant<x == 0>;
01141
01142
01144
                  static std::string to_string() {
01145
                     return std::to_string(x);
01146
01147
              };
01148
01151
              template<auto x>
01152
              using inject_constant_t = val<static_cast<int64_t>(x)>;
01153
01158
              template<typename v>
01159
              using inject_ring_t = v;
01160
01162
              using zero = val<0>:
             using one = val<1>;
01164
01166
              static constexpr bool is_field = false;
01168
              static constexpr bool is_euclidean_domain = true;
01169
          private:
01170
              template<typename v1, typename v2>
01171
              struct add {
01172
01173
                 using type = val<v1::v + v2::v>;
01174
01175
01176
              template<typename v1, typename v2>
01177
              struct sub {
                 using type = val<v1::v - v2::v>;
01178
01179
01180
01181
              template<typename v1, typename v2>
01182
              struct mul {
                 using type = val<v1::v* v2::v>;
01183
01184
01185
01186
              template<typename v1, typename v2>
01187
              struct div {
                 using type = val<v1::v / v2::v>;
01188
01189
01190
              template<typename v1, typename v2>
01191
01192
              struct remainder {
01193
                 using type = val<v1::v% v2::v>;
01194
01195
              template<typename v1, typename v2>
01196
01197
              struct at {
01198
                 using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01199
01200
01201
              template<typename v1, typename v2>
01202
              struct lt {
                 using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01203
01204
01205
01206
              template<typename v1, typename v2>
01207
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01208
01209
01210
01211
              template<typename v>
01212
              struct pos {
01213
                 using type = std::bool_constant<(v::v > 0)>;
01214
              };
01215
01216
          public:
              template<typename v1, typename v2>
01221
              using add_t = typename add<v1, v2>::type;
01222
01226
              template<typename v1, typename v2> \,
01227
              using sub_t = typename sub<v1, v2>::type;
01228
```

```
01232
              template<typename v1, typename v2>
01233
              using mul_t = typename mul<v1, v2>::type;
01234
01239
              template<typename v1, typename v2>
01240
              using div t = typename div<v1, v2>::type;
01241
01245
              template<typename v1, typename v2>
01246
              using mod_t = typename remainder<v1, v2>::type;
01247
01252
              template<typename v1, typename v2>
01253
              using gt_t = typename gt<v1, v2>::type;
01254
01259
              template<typename v1, typename v2>
01260
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01261
01266
              template<typename v1, typename v2>
01267
              using lt_t = typename lt<v1, v2>::type;
01268
              template<typename v1, typename v2>
01274
              static constexpr bool lt_v = lt_t<v1, v2>::value;
01275
01280
              template<typename v1, typename v2>
01281
              using eq_t = typename eq<v1, v2>::type;
01282
01287
              template<typename v1, typename v2>
01288
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01289
01294
              template<typename v1, typename v2>
01295
              using gcd_t = gcd_t < i64, v1, v2>;
01296
01300
              template < typename v >
01301
              using pos_t = typename pos<v>::type;
01302
01306
              template<typename v>
01307
              static constexpr bool pos_v = pos_t<v>::value;
         };
01308
01309
01311
          template<>
01312
          struct Embed<i32, i64> {
01315
           template<typename val>
01316
              using type = i64::val<static_cast<int64_t>(val::v)>;
01317
01318 } // namespace aerobus
01319
01320 // z/pz
01321 namespace aerobus {
01327
         template<int32_t p>
01328
         struct zpz {
01330
             using inner_type = int32_t;
01331
01334
              template<int32_t x>
01335
              struct val {
01337
                  using enclosing_type = zpz;
01339
                  static constexpr int32_t v = x % p;
01340
01343
                  template<typename valueType>
01344
                  static constexpr INLINED DEVICE valueType get() {
01345
                      return static_cast<valueType>(x % p);
01346
01347
01349
                  using is zero t = std::bool constant<v == 0>;
01350
01352
                  static constexpr bool is_zero_v = v == 0;
01353
01356
                  static std::string to_string() {
01357
                     return std::to_string(x % p);
01358
01359
              };
01360
01363
              template<auto x>
01364
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01365
01367
             using zero = val<0>;
01368
01370
             using one = val<1>;
01371
01373
              static constexpr bool is_field = is_prime::value;
01374
01376
              static constexpr bool is_euclidean_domain = true;
01377
01378
           private:
01379
              template<typename v1, typename v2>
01380
01381
                  using type = val<(v1::v + v2::v) % p>;
01382
              };
01383
01384
              template<tvpename v1, tvpename v2>
```

```
01385
                          struct sub {
                                using type = val<(v1::v - v2::v) % p>;
01386
01387
                          };
01388
                          template<typename v1, typename v2>
01389
01390
                          struct mul {
01391
                                 using type = val<(v1::v* v2::v) % p>;
01392
01393
01394
                          template<typename v1, typename v2>
01395
                          struct div {
                               using type = val<(v1::v% p) / (v2::v % p)>;
01396
01397
01398
01399
                          template<typename v1, typename v2>
01400
                          struct remainder {
                                 using type = val<(v1::v% v2::v) % p>;
01401
01402
01403
01404
                          template<typename v1, typename v2>
01405
01406
                                 using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01407
01408
01409
                          template<typename v1, typename v2>
01410
                          struct lt {
01411
                                   \  \  \, using \ type = std::conditional\_t < (v1::v% \ p < v2::v% \ p), \ std::true\_type, \ std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type>; \\ \  \  \, using \ type = std::false\_type = std::false\_typ
01412
01413
01414
                          template<typename v1, typename v2>
01415
                          struct eq {
01416
                                 using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01417
01418
01419
                          template<typename v1>
01420
                          struct pos {
                                 using type = std::bool_constant<(v1::v > 0)>;
01421
01423
01424
                    public:
01428
                          template<typename v1, typename v2> ^{\circ}
01429
                          using add_t = typename add<v1, v2>::type;
01430
01434
                          template<typename v1, typename v2>
01435
                         using sub_t = typename sub<v1, v2>::type;
01436
01440
                          template<typename v1, typename v2>
01441
                          using mul_t = typename mul<v1, v2>::type;
01442
01446
                          template<typename v1, typename v2>
01447
                          using div_t = typename div<v1, v2>::type;
01448
01452
                          template<typename v1, typename v2>
01453
                          using mod_t = typename remainder<v1, v2>::type;
01454
01458
                          template<typename v1, typename v2>
                          using gt_t = typename gt<v1, v2>::type;
01460
01464
                          template<typename v1, typename v2>
01465
                          static constexpr bool gt_v = gt_t<v1, v2>::value;
01466
01470
                          template<typename v1, typename v2>
01471
                          using lt_t = typename lt<v1, v2>::type;
01472
01476
                          template<typename v1, typename v2>
01477
                          static constexpr bool lt_v = lt_t<v1, v2>::value;
01478
                          template<typename v1, typename v2>
01482
01483
                          using eg t = typename eg<v1, v2>::type;
01484
01488
                          template<typename v1, typename v2>
01489
                          static constexpr bool eq_v = eq_t<v1, v2>::value;
01490
01494
                          template<typename v1, typename v2> ^{\circ}
                          using gcd_t = gcd_t<i32, v1, v2>;
01495
01496
01499
                          template<typename v1>
01500
                          using pos_t = typename pos<v1>::type;
01501
01504
                          template<tvpename v>
01505
                          static constexpr bool pos_v = pos_t<v>::value;
01506
                  };
01507
01510
                  template<int32_t x>
01511
                   struct Embed<zpz<x>, i32> {
                          template <typename val>
using type = i32::val<val::v>;
01514
01515
```

```
01517 } // namespace aerobus
01518
01519 // polynomial
template<typename Ring>
                 requires IsEuclideanDomain<Ring>
01527
01528
                 struct polynomial {
01529
                        static constexpr bool is_field = false;
                        static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01530
01531
01534
                         template<typename P>
01535
                         struct horner_reduction_t {
01536
                               template<size_t index, size_t stop>
01537
                                struct inner {
01538
                                       template<typename accum, typename x>
                                       using type = typename horner_reduction_t<P>::template inner<index + 1, stop>
01539
01540
                                              ::template type<
01541
                                                    typename Ring::template add_t<
01542
                                                            typename Ring::template mul_t<x, accum>,
01543
                                                            typename P::template coeff_at_t<P::degree - index>
01544
                                                     >, x>;
01545
                               };
01546
01547
                                template<size_t stop>
                                struct inner<stop, stop> {
01548
01549
                                       template<typename accum, typename x>
01550
                                       using type = accum;
01551
                               };
01552
                        };
01553
01557
                         template<typename coeffN, typename... coeffs>
01558
01560
                                using ring_type = Ring;
                               using enclosing_type = polynomial<Ring>;
static constexpr size_t degree = sizeof...(coeffs);
01562
01564
01566
                               using aN = coeffN;
01568
                               using strip = val<coeffs...>;
01570
                                using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01572
                                static constexpr bool is_zero_v = is_zero_t::value;
01573
01574
                          private:
01575
                               template<size_t index, typename E = void>
01576
                               struct coeff_at {};
01577
01578
                                template<size_t index>
                                \verb|struct coeff_at<index|, \verb|std::enable_if_t<(index|>= 0 && index|<= sizeof...(coeffs))|| >> 0 && index|| >>
01579
                                       using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01580
01581
01582
01583
                                template<size_t index>
01584
                                \verb|struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs)) > (
01585
                                      using type = typename Ring::zero;
01586
                                };
01587
01588
                          public:
01591
                               template<size_t index>
01592
                                using coeff_at_t = typename coeff_at<index>::type;
01593
01596
                                static std::string to string() {
01597
                                       return string_helper<coeffN, coeffs...>::func();
01598
01599
01604
                                template<typename arithmeticType>
01605
                                static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01606
                                      #ifdef WITH CUDA FP16
01607
                                       arithmeticTvpe start:
01608
                                       if constexpr (std::is same v<arithmeticType, half2>) {
                                             start = \underline{\quad }half2(0, 0);
01610
01611
                                              start = static_cast<arithmeticType>(0);
01612
                                       #else
01613
                                       arithmeticType start = static cast<arithmeticType>(0);
01614
01615
                                       #endif
01616
                                       return horner_evaluation<arithmeticType, val>
01617
                                                    ::template inner<0, degree + 1>
01618
                                                     ::func(start, x);
01619
                               }
01620
01621
                                template<typename x>
                                using value_at_t = horner_reduction_t<val>
01622
01623
                                        ::template inner<0, degree + 1>
01624
                                       ::template type<typename Ring::zero, x>;
01625
                        };
01626
```

```
template<typename coeffN>
              struct val<coeffN> {
01630
                  using ring_type = Ring;
01632
01634
                  using enclosing_type = polynomial<Ring>;
01636
                  static constexpr size t degree = 0;
                  using aN = coeffN;
01637
                  using strip = val<coeffN>;
01638
01639
                  using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01640
01641
                  static constexpr bool is_zero_v = is_zero_t::value;
01642
                  template<size_t index, typename E = void>
01643
01644
                  struct coeff_at {};
01645
01646
                  template<size_t index>
01647
                  struct coeff_at<index, std::enable_if_t<(index == 0)» {</pre>
01648
                      using type = aN;
01649
01650
01651
                  template<size_t index>
01652
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01653
                      using type = typename Ring::zero;
01654
01655
01656
                  template<size_t index>
01657
                  using coeff_at_t = typename coeff_at<index>::type;
01658
01659
                  static std::string to_string() {
01660
                      return string_helper<coeffN>::func();
01661
01662
01663
                  template<typename arithmeticType>
01664
                  static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01665
                      return coeffN::template get<arithmeticType>();
01666
01667
                  template<typename x>
01668
01669
                  using value_at_t = coeffN;
01670
              };
01671
01673
              using zero = val<typename Ring::zero>;
              using one = val<typename Ring::one>;
01675
              using X = val<typename Ring::one, typename Ring::zero>;
01677
01678
01679
01680
              template<typename P, typename E = void>
01681
              struct simplify;
01682
              template <typename P1, typename P2, typename I>
01683
01684
              struct add low:
01685
01686
              template<typename P1, typename P2>
01687
              struct add {
01688
                  using type = typename simplify<typename add_low<
01689
                  P1.
01690
                  P2,
01691
                  internal::make_index_sequence_reverse<
01692
                  std::max(P1::degree, P2::degree) + 1
01693
                  »::type>::type;
01694
              };
01695
01696
              template <typename P1, typename P2, typename I>
01697
              struct sub_low;
01698
01699
              template <typename P1, typename P2, typename I>
01700
              struct mul_low;
01701
01702
              template<tvpename v1, tvpename v2>
01703
              struct mul {
01704
                      using type = typename mul_low<
01705
01706
                          v2,
01707
                          internal::make_index_sequence_reverse<</pre>
                          v1::degree + v2::degree + 1
01708
01709
                          »::type;
01710
              };
01711
01712
              template<typename coeff, size_t deg>
01713
              struct monomial;
01714
01715
              template<typename v, typename E = void>
              struct derive_helper {};
01717
01718
              template<typename v>
01719
              struct derive_helper<v, std::enable_if_t<v::degree == 0» {</pre>
01720
                  using type = zero;
01721
              };
```

```
01722
01723
              template<typename v>
01724
              struct derive_helper<v, std::enable_if_t<v::degree != 0» {
01725
                  using type = typename add<
01726
                      typename derive_helper<typename simplify<typename v::strip>::type>::type,
01727
                      typename monomial<
01728
                          typename Ring::template mul_t<</pre>
01729
                              typename v::aN,
01730
                              typename Ring::template inject_constant_t<(v::degree)>
01731
01732
                          v::degree - 1
                      >::type
01733
01734
                  >::type;
01735
              };
01736
01737
              template<typename v1, typename v2, typename E = void>
01738
              struct eq_helper {};
01739
01740
              template<typename v1, typename v2>
01741
              struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
01742
                 using type = std::false_type;
01743
01744
01745
01746
              template<typename v1, typename v2>
01747
              struct eq_helper<v1, v2, std::enable_if_t<
01748
                  v1::degree == v2::degree &&
01749
                  (v1::degree != 0 || v2::degree != 0) &&
01750
                  std::is_same<
01751
                  typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
                  std::false_type
01752
01753
                  >::value
01754
01755
              > {
01756
                  using type = std::false_type;
01757
              };
01758
01759
              template<typename v1, typename v2>
01760
              struct eq_helper<v1, v2, std::enable_if_t<
01761
                v1::degree == v2::degree &&
01762
                  (v1::degree != 0 || v2::degree != 0) &&
01763
                  std::is_same<
01764
                  typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01765
                  std::true_type
01766
                  >::value
01767
              » {
01768
                  using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01769
              } ;
01770
01771
              template<typename v1, typename v2> ^{\circ}
              struct eq_helper<v1, v2, std::enable_if_t<
01772
01773
                  v1::degree == v2::degree &&
01774
                  (v1::degree == 0)
01775
01776
                  using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01777
              };
01778
01779
              template<typename v1, typename v2, typename E = void>
01780
              struct lt_helper {};
01781
              template<typename v1, typename v2>
struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {</pre>
01782
01783
01784
                  using type = std::true_type;
01785
01786
01787
              template<typename v1, typename v2>
01788
              struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01789
                  using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01790
01791
              template<typename v1, typename v2>
struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
01792
01793
01794
                  using type = std::false_type;
01795
01796
01797
              template<typename v1, typename v2, typename E = void>
01798
              struct gt_helper {};
01799
              01800
01801
                  using type = std::true_type;
01802
01803
01804
01805
              template<typename v1, typename v2>
01806
              struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01807
                  using type = std::false_type;
01808
              };
```

```
01809
              template<typename v1, typename v2>
struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {</pre>
01810
01811
01812
                 using type = std::false_type;
01813
01814
01815
              // when high power is zero : strip
01816
              template<typename P>
01817
              struct simplify<P, std::enable_if_t<
01818
                  std::is_same<
                  typename Ring::zero,
01819
                  typename P::aN
01820
01821
                  >::value && (P::degree > 0)
01822
01823
                  using type = typename simplify<typename P::strip>::type;
01824
01825
01826
              // otherwise : do nothing
01827
              template<typename P>
              struct simplify<P, std::enable_if_t<
01828
01829
                  !std::is_same<
01830
                  typename Ring::zero,
01831
                  typename P::aN
01832
                  >::value && (P::degree > 0)
01833
              » {
01834
                  using type = P;
01835
              } ;
01836
              // do not simplify constants
01837
01838
              template<typename P>
01839
              struct simplify<P, std::enable if t<P::degree == 0» {
01840
                  using type = P;
01841
01842
01843
              // addition at
              template<typename P1, typename P2, size_t index>
01844
01845
              struct add at {
01846
                  using type =
01847
                       typename Ring::template add_t<
01848
                           typename P1::template coeff_at_t<index>,
01849
                           typename P2::template coeff_at_t<index»;</pre>
01850
              };
01851
01852
              template<typename P1, typename P2, size_t index>
              using add_at_t = typename add_at<P1, P2, index>::type;
01853
01854
01855
              template<typename P1, typename P2, std::size_t... I>
01856
              struct add_low<P1, P2, std::index_sequence<I...» {
                  using type = val<add_at_t<P1, P2, I>...>;
01857
01858
01859
01860
              // substraction at
01861
              template<typename P1, typename P2, size_t index>
01862
              struct sub_at {
01863
                  using type =
01864
                       typename Ring::template sub t<
01865
                          typename P1::template coeff_at_t<index>,
01866
                           typename P2::template coeff_at_t<index>;
01867
01868
              template<typename P1, typename P2, size_t index>
01869
01870
              using sub_at_t = typename sub_at<P1, P2, index>::type;
01871
01872
              template<typename P1, typename P2, std::size_t... I>
01873
              struct sub_low<P1, P2, std::index_sequence<I...» {
01874
                  using type = val<sub_at_t<P1, P2, I>...>;
01875
01876
01877
              template<typename P1, typename P2>
01878
              struct sub {
01879
                  using type = typename simplify<typename sub_low<
                  P1,
01880
01881
                  P2,
01882
                  internal::make_index_sequence_reverse<</pre>
                  std::max(P1::degree, P2::degree) + 1
01883
01884
                  »::type>::type;
01885
01886
              // multiplication at
01887
01888
              template<typename v1, typename v2, size t k, size t index, size t stop>
01889
              struct mul_at_loop_helper {
                  using type = typename Ring::template add_t<
01890
01891
                       typename Ring::template mul_t<</pre>
01892
                       typename v1::template coeff_at_t<index>,
01893
                       typename v2::template coeff_at_t<k - index>
01894
01895
                       typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
```

```
01896
                  >;
01897
01898
01899
              template<typename v1, typename v2, size_t k, size_t stop>
              struct mul_at_loop_helper<v1, v2, k, stop, stop> {
    using type = typename Ring::template mul_t<</pre>
01900
01901
                      typename v1::template coeff_at_t<stop>,
01902
01903
                       typename v2::template coeff_at_t<0»;
01904
              };
01905
01906
              template <typename v1, typename v2, size_t k, typename E = void>
01907
              struct mul at {}:
01908
              01909
01910
01911
                 using type = typename Ring::zero;
01912
              };
01913
01914
              template<typename v1, typename v2, size_t k>
              struct mul_at < v1, v2, k, std::enable_if_t < (k >= 0) && (k <= v1::degree + v2::degree) <math>>  {
01915
01916
                 using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01917
01918
              template<typename P1, typename P2, size_t index>
using mul_at_t = typename mul_at<P1, P2, index>::type;
01919
01920
01921
               template<typename P1, typename P2, std::size_t... I>
01922
01923
              struct mul_low<P1, P2, std::index_sequence<I...» {</pre>
01924
                  using type = val<mul_at_t<P1, P2, I>...>;
01925
01926
01927
              // division helper
01928
               template< typename A, typename B, typename Q, typename R, typename E = void>
01929
              struct div_helper {};
01930
01931
              template<typename A, typename B, typename Q, typename R>
01932
              struct div_helper<A, B, Q, R, std::enable_if_t<
                   (R::degree < B::degree) ||
01933
01934
                   (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
01935
                   using q_type = Q;
01936
                  using mod_type = R;
01937
                  using gcd_type = B;
01938
              }:
01939
              template<typename A, typename B, typename Q, typename R> struct div_helper<A, B, Q, R, std::enable_if_t<
01940
01941
01942
                   (R::degree >= B::degree) &&
01943
                   !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
               private: // NOLINT
01944
                  using rN = typename R::aN;
01945
01946
                  using bN = typename B::aN;
                   using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
01947
     B::degree>::type;
01948
                 using rr = typename sub<R, typename mul<pT, B>::type>::type;
                  using qq = typename add<Q, pT>::type;
01949
01950
01951
01952
                  using q_type = typename div_helper<A, B, qq, rr>::q_type;
01953
                   using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
01954
                  using gcd_type = rr;
01955
              }:
01956
01957
              template<typename A, typename B>
01958
01959
                   static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
                  using q_type = typename div_helper<A, B, zero, A>::q_type; using m_type = typename div_helper<A, B, zero, A>::mod_type;
01960
01961
01962
              };
01963
              template<typename P>
01965
              struct make_unit {
01966
                  using type = typename div<P, val<typename P::aN»::q_type;
01967
01968
01969
              template<typename coeff, size_t deg>
01970
              struct monomial {
01971
                  using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
01972
01973
01974
              template<typename coeff>
01975
              struct monomial<coeff, 0> {
01976
                  using type = val<coeff>;
01977
01978
01979
              template<typename arithmeticType, typename P>
01980
              struct horner evaluation {
01981
                  template<size t index, size t stop>
```

```
struct inner {
01983
                      static constexpr DEVICE INLINED arithmeticType func(
                           const arithmeticType& accum, const arithmeticType& x) {
  return horner_evaluation<arithmeticType, P>::template inner<index + 1,</pre>
01984
01985
      stop>::func(
01986
                                internal::fma helper<arithmeticTvpe>::eval(
01987
01988
                                    accum,
get<arithmeticType>()), x);
01990
01989
                                    P::template coeff_at_t<P::degree - index>::template
01991
                   };
01992
01993
                   template<size_t stop>
01994
                   struct inner<stop, stop> {
                       static constexpr DEVICE INLINED arithmeticType func(
01995
01996
                           const arithmeticType& accum, const arithmeticType& x) {
01997
                           return accum;
01998
01999
                   };
02000
              };
02001
02002
               template<typename coeff, typename... coeffs>
02003
               struct string helper {
02004
                   static std::string func() {
02005
                      std::string tail = string_helper<coeffs...>::func();
02006
                       std::string result = "";
02007
                       if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
                       return tail;
} else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
02008
02009
                           if (sizeof...(coeffs) == 1) {
    result += "x";
02010
02011
02012
                            } else {
                                result += "x^" + std::to_string(sizeof...(coeffs));
02013
02014
02015
                       } else {
02016
                           if (sizeof...(coeffs) == 1) {
02017
                                result += coeff::to_string() + " x";
02018
                               result += coeff::to_string()
+ " x^" + std::to_string(sizeof...(coeffs));
02019
02020
02021
                            }
02022
                       }
02023
02024
                        if (!tail.empty()) {
                            if (tail.at(0) != '-') {
02025
                                result += " + " + tail;
02026
02027
                            } else {
                                result += " - " + tail.substr(1);
02028
02029
02030
02031
02032
                       return result;
02033
                   }
02034
              };
02035
02036
               template<typename coeff>
               struct string_helper<coeff> {
02037
02038
                  static std::string func() {
02039
                       if (!std::is_same<coeff, typename Ring::zero>::value) {
02040
                           return coeff::to_string();
02041
                       } else {
                           return "";
02042
02043
02044
                   }
02045
              };
02046
02047
           public:
02050
              template<typename P>
02051
              using simplify_t = typename simplify<P>::type;
02052
02056
               template<typename v1, typename v2>
02057
               using add_t = typename add<v1, v2>::type;
02058
02062
               template<typename v1, typename v2>
02063
               using sub_t = typename sub<v1, v2>::type;
02064
02068
               template<typename v1, typename v2>
02069
               using mul_t = typename mul<v1, v2>::type;
02070
02074
               template<typename v1, typename v2>
02075
               using eq_t = typename eq_helper<v1, v2>::type;
02076
02080
               template<typename v1, typename v2>
02081
               using lt_t = typename lt_helper<v1, v2>::type;
02082
02086
               template<tvpename v1, tvpename v2>
```

```
using gt_t = typename gt_helper<v1, v2>::type;
02088
02092
              template<typename v1, typename v2>
02093
              using div_t = typename div<v1, v2>::q_type;
02094
02098
              template<typename v1, typename v2>
02099
              using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
02100
02104
              template<typename coeff, size_t deg>
02105
              using monomial_t = typename monomial<coeff, deg>::type;
02106
02109
              template<tvpename v>
02110
              using derive_t = typename derive_helper<v>::type;
02111
02114
              template<typename v>
02115
              using pos_t = typename Ring::template pos_t<typename v::aN>;
02116
02119
              template<typename v>
02120
              static constexpr bool pos_v = pos_t<v>::value;
02121
02125
              template<typename v1, typename v2>
              using gcd_t = std::conditional_t<
02126
02127
                  Ring::is_euclidean_domain,
                  typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
02128
02129
                  void>;
02130
02133
              template<auto x>
02134
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
02135
02138
              template<typename v>
02139
              using inject ring t = val<v>:
02140
          };
02141 } // namespace aerobus
02142
02143 // fraction field
02144 namespace aerobus {
02145
         namespace internal {
             template<typename Ring, typename E = void>
02147
              requires IsEuclideanDomain<Ring>
02148
              struct _FractionField {};
02149
02150
              template<typename Ring>
              requires IsEuclideanDomain<Ring>
02151
              struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain> {
    static constexpr bool is_field = true;
02152
02154
02155
                  static constexpr bool is_euclidean_domain = true;
02156
02157
               private:
                  template<typename val1, typename val2, typename E = void>
02158
02159
                  struct to_string_helper {};
02160
02161
                  template<typename val1, typename val2>
02162
                  struct to_string_helper <val1, val2,
02163
                      std::enable_if_t<
02164
                      Ring::template eq_t<
02165
                      val2, typename Ring::one
02166
                      >::value
02167
02168
02169
                      static std::string func() {
02170
                          return val1::to_string();
02171
02172
                  };
02173
02174
                  template<typename val1, typename val2>
02175
                  struct to_string_helper<val1, val2,
02176
                      std::enable if t<
02177
                      !Ring::template eq_t<
02178
                      val2,
02179
                      typename Ring::one
02180
                      >::value
02181
02182
                  > {
02183
                      static std::string func() {
02184
                          return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
02185
02186
                  };
02187
               public:
02188
02192
                  template<typename vall, typename val2>
02193
                  struct val {
                      using x = val1;
02195
                      using y = val2;
02197
02199
                      using is_zero_t = typename val1::is_zero_t;
02201
                      static constexpr bool is_zero_v = val1::is_zero_t::value;
02202
02204
                      using ring type = Ring;
```

```
using enclosing_type = _FractionField<Ring>;
02206
02209
                        static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
02210
02214
                       template<typename valueType>
02215
                       static constexpr INLINED DEVICE valueType get() {
                           return internal::staticcast<valueType, typename ring_type::inner_type>::template
02216
      func<x::v>() /
02217
                                internal::staticcast<valueType, typename ring_type::inner_type>::template
      func<y::v>();
02218
                       }
02219
02222
                       static std::string to string() {
02223
                           return to_string_helper<val1, val2>::func();
02224
02225
02230
                       template<typename arithmeticType>
                       static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& v) {
    return x::eval(v) / y::eval(v);
02231
02232
02233
02234
                  };
02235
                  using zero = val<typename Ring::zero, typename Ring::one>;
using one = val<typename Ring::one, typename Ring::one>;
02237
02239
02240
02243
                  template<typename v>
02244
                  using inject_t = val<v, typename Ring::one>;
02245
02248
                  template<auto x>
                  using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
02249
     Ring::one>;
02250
02253
02254
                  using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
02255
02257
                  using ring_type = Ring;
02258
02259
               private:
02260
                  template<typename v, typename E = void>
02261
                  struct simplify {};
02262
02263
                   // x = 0
                  template<typename v>
02264
02265
                  struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {
02266
                      using type = typename _FractionField<Ring>::zero;
02267
02268
                  // x != 0
02269
02270
                  template<tvpename v>
                  struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value» {</pre>
02271
02272
                   private:
02273
                       using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
                       using newx = typename Ring::template div_t<typename v::x, _gcd>;
02274
02275
                      using newy = typename Ring::template div_t<typename v::y, _gcd>;
02276
02277
                       using posx = std::conditional t<
02278
                                            !Ring::template pos_v<newy>,
02279
                                            typename Ring::template sub_t<typename Ring::zero, newx>,
02280
                                            newx>;
02281
                       using posy = std::conditional_t<
02282
                                            !Ring::template pos_v<newy>,
02283
                                            typename Ring::template sub_t<typename Ring::zero, newy>,
02284
                                            newy>;
02285
                   public:
02286
                       using type = typename _FractionField<Ring>::template val<posx, posy>;
02287
                  };
02288
02289
               public:
02292
                  template<typename v>
02293
                  using simplify_t = typename simplify<v>::type;
02294
02295
02296
                  template<typename v1, typename v2>
02297
                   struct add {
02298
                   private:
02299
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02300
                       using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02301
                       using dividend = typename Ring::template add_t<a, b>;
                       using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02302
02303
                      using g = typename Ring::template gcd_t<dividend, diviser>;
02304
                   public:
02305
                       using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
02306
      diviser»;
02307
02308
02309
                  template<tvpename v>
```

```
struct pos {
                      using type = std::conditional_t<
02311
02312
                            (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
02313
                            (!Ring::template pos_v < typename v::x > \&\& !Ring::template pos_v < typename v::y >) \textit{,} \\
02314
                           std::true_type,
02315
                           std::false type>;
02316
                  };
02317
02318
                  template<typename v1, typename v2>
                   struct sub {
02319
02320
                   private:
02321
                      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>;
02322
02323
                       using dividend = typename Ring::template sub_t<a, b>;
02324
                       using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02325
                       using g = typename Ring::template gcd_t<dividend, diviser>;
02326
02327
                   public:
02328
                      using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
     diviser»;
02329
02330
02331
                  template<typename v1, typename v2>
02332
                   struct mul {
02333
                   private:
02334
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
                       using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02335
02336
                   public:
02337
02338
                      using type = typename _FractionField<Ring>::template simplify t<val<a, b>;
02339
                  };
02340
02341
                  template<typename v1, typename v2, typename E = void>
02342
                   struct div {};
02343
                  template<typename v1, typename v2>
struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename</pre>
02344
02345
      _FractionField<Ring>::zero>::value» {
02346
                   private:
02347
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02348
                       using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02349
                   public:
02350
02351
                      using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
02352
                  };
02353
02354
                  template<typename v1, typename v2>
02355
                   struct div<v1, v2, std::enable_if_t<
                       std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
02356
02357
                       using type = one;
02358
                  };
02359
02360
                  template<typename v1, typename v2>
02361
                  struct eq {
02362
                       using type = std::conditional_t<
                               std::is_same<typename simplify_t<vl>::x, typename simplify_t<v2>::x>::value &&
02363
02364
                               std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
02365
                           std::true_type,
02366
                           std::false_type>;
02367
                  };
02368
02369
                  template<typename v1, typename v2, typename E = void>
02370
                  struct qt;
02371
02372
                  template<typename v1, typename v2>
02373
                   struct gt<v1, v2, std::enable_if_t<
02374
                      (eq<v1, v2>::type::value)
02375
02376
                       using type = std::false_type;
02377
                  };
02378
02379
                   template<typename v1, typename v2>
                  struct gt<v1, v2, std::enable_if_t<
     (!eq<v1, v2>::type::value) &&
02380
02381
02382
                       (!pos<v1>::type::value) && (!pos<v2>::type::value)
02383
02384
                       using type = typename gt<
02385
                           typename sub<zero, v1>::type, typename sub<zero, v2>::type
                       >::type;
02386
02387
                  }:
02388
02389
                   template<typename v1, typename v2>
02390
                   struct gt<v1, v2, std::enable_if_t<
02391
                       (!eq<v1, v2>::type::value) &&
02392
                       (pos<v1>::type::value) && (!pos<v2>::type::value)
02393
02394
                       using type = std::true type;
```

```
02395
                   };
02396
02397
                   template<typename v1, typename v2>
                   struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02398
02399
                       (!pos<v1>::type::value) && (pos<v2>::type::value)
02400
02401
02402
                       using type = std::false_type;
02403
                   };
02404
02405
                   template<typename v1, typename v2>
                   struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02406
02407
02408
                       (pos<v1>::type::value) && (pos<v2>::type::value)
02409
02410
                       using type = typename Ring::template gt_t<
                           typename Ring::template mul_t<v1::x, v2::y>,
typename Ring::template mul_t<v2::y, v2::x>
02411
02412
02413
02414
                   } ;
02415
02416
                public:
                   template<typename v1, typename v2>
02420
02421
                   using add_t = typename add<v1, v2>::type;
02422
                   template<typename v1, typename v2>
02427
02428
                   using mod_t = zero;
02429
02434
                   template<typename v1, typename v2>
02435
                   using gcd_t = v1;
02436
02440
                   template<typename v1, typename v2>
02441
                   using sub_t = typename sub<v1, v2>::type;
02442
02446
                   template<typename v1, typename v2>
                   using mul_t = typename mul<v1, v2>::type;
02447
02448
02452
                   template<typename v1, typename v2>
02453
                   using div_t = typename div<v1, v2>::type;
02454
02458
                   template<typename v1, typename v2>
02459
                   using eq_t = typename eq<v1, v2>::type;
02460
02464
                   template<typename v1, typename v2>
02465
                   static constexpr bool eq_v = eq<v1, v2>::type::value;
02466
02470
                   template<typename v1, typename v2>
02471
                   using gt_t = typename gt<v1, v2>::type;
02472
02476
                   template<typename v1, typename v2>
                   static constexpr bool gt_v = gt<v1, v2>::type::value;
02478
02481
                   template<typename v1>
02482
                   using pos_t = typename pos<v1>::type;
02483
02486
                   template<typename v>
02487
                   static constexpr bool pos_v = pos_t < v > :: value;
02488
               };
02489
02490
               template<typename Ring, typename E = void>
02491
               requires IsEuclideanDomain<Ring>
               struct FractionFieldImpl {};
02492
02493
02494
               // fraction field of a field is the field itself
02495
               template<typename Field>
02496
               requires IsEuclideanDomain<Field>
               struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {</pre>
02497
02498
                   using type = Field;
02499
                   template<typename v>
02500
                   using inject_t = v;
02501
              };
02502
02503
               // fraction field of a ring is the actual fraction field
02504
               template<typename Ring>
02505
               requires IsEuclideanDomain<Ring>
02506
               struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {</pre>
02507
                   using type = _FractionField<Ring>;
02508
          } // namespace internal
02509
02510
02513
          template<typename Ring>
          requires IsEuclideanDomain<Ring>
02515
          using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02516
02519
          template<typename Ring>
          struct Embed<Ring, FractionField<Ring» {</pre>
02520
02523
               template<typename v>
```

```
using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02525
02526 }
         // namespace aerobus
02527
02528
02529 // short names for common types
02530 namespace aerobus {
02534
          template<typename X, typename Y>
02535
          requires IsRing<typename X::enclosing_type> &&
02536
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02537
          using add_t = typename X::enclosing_type::template add_t<X, Y>;
02538
02542
          template<typename X, typename Y>
02543
          requires IsRing<typename X::enclosing_type> &&
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02544
02545
          using sub_t = typename X::enclosing_type::template sub_t<X, Y>;
02546
02550
          template<typename X, typename Y>
          requires IsRing<typename X::enclosing_type> &&
02551
02552
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02553
          using mul_t = typename X::enclosing_type::template mul_t<X, Y>;
02554
02558
          template<typename X, typename Y>
          requires IsEuclideanDomain<typename X::enclosing_type> &&
   (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02559
02560
          using div_t = typename X::enclosing_type::template div_t<X, Y>;
02561
02562
02565
          using q32 = FractionField<i32>;
02566
02569
          using fpq32 = FractionField<polynomial<q32>>;
02570
02573
          using q64 = FractionField<i64>;
02574
02576
          using pi64 = polynomial<i64>;
02577
02579
          using pg64 = polynomial<g64>;
02580
02582
          using fpq64 = FractionField<polynomial<q64>>;
02583
02588
          template<typename Ring, typename v1, typename v2>
02589
          using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02590
02597
          template<typename v>
02598
          using embed_int_poly_in_fractions_t =
02599
                   typename Embed<
02600
                      polynomial<typename v::ring_type>,
02601
                      polynomial<FractionField<typename v::ring_type>»::template type<v>;
02602
          template<int64_t p, int64_t q>
02606
          using make_q64_t = typename q64::template simplify_t<
02607
02608
                       typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02609
02613
          template<int32_t p, int32_t q>
02614
          using make_q32_t = typename q32::template simplify_t<
                       typename q32::val<i32::inject_constant_t<p>, i32::inject_constant t<q>>>;
02615
02616
02621
          template<typename Ring, typename v1, typename v2>
02622
          using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02627
          template<typename Ring, typename v1, typename v2>
02628
          using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02629
02631
          template<>
02632
          struct Embed<q32, q64> {
02635
            template<typename v>
02636
              using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02637
02638
          template<typename Small, typename Large>
02642
02643
          struct Embed<polynomial<Small>, polynomial<Large» {
02644
          private:
02645
              template<typename v, typename i>
02646
              struct at_low;
02647
02648
              template<typename v, size_t i>
02649
              struct at_index {
                  using type = typename Embed<Small, Large>::template
     type<typename v::template coeff_at_t<i>>;
02651
02652
02653
              template<typename v. size t... Is>
              struct at_low<v, std::index_sequence<Is...» {
   using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
02654
02655
02656
02657
02658
           public:
02661
              template<typename v>
02662
              using type = typename at low<v, typename internal::make index sequence reverse<v::degree +
```

```
1»::type;
02663
02664
02668
          template<typename Ring, auto... xs>
          using make_int_polynomial_t = typename polynomial<Ring>::template val<</pre>
02669
                  typename Ring::template inject_constant_t<xs>...>;
02670
02671
02675
          template<typename Ring, auto... xs>
02676
          using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<</pre>
02677
                  typename FractionField<Ring>::template inject_constant_t<xs>...>;
02678 } // namespace aerobus
02679
02680 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02681 namespace aerobus {
02682
        namespace internal {
02683
             template<typename T, size_t x, typename E = void>
02684
              struct factorial {}:
02685
02686
              template<typename T, size_t x>
              struct factorial<T, x, std::enable_if_t<(x > 0)» {
02688
              private:
02689
                  template<typename, size_t, typename>
02690
                  friend struct factorial;
02691
              public:
02692
                  using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
      x - 1>::type>;
02693
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02694
              };
02695
02696
             template<tvpename T>
02697
              struct factorial<T, 0> {
02698
              public:
02699
                  using type = typename T::one;
02700
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02701
              };
          } // namespace internal
02703
02707
          template<typename T, size_t i>
02708
          using factorial_t = typename internal::factorial<T, i>::type;
02709
          template<typename T, size_t i>
inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02713
02714
02715
02716
02717
              template<typename T, size_t k, size_t n, typename E = void>
02718
              struct combination_helper {};
02719
02720
              template<typename T, size t k, size t n>
02721
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
02722
                  using type = typename FractionField<T>::template mul_t<</pre>
                       typename combination_helper<T, k - 1, n - 1>::type,
02723
02724
                       makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02725
              };
02726
02727
              template<typename T, size_t k, size_t n>
02728
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0) \times {
02729
                  using type = typename combination_helper<T, n - k, n>::type;
02730
02731
              template<typename T, size_t n>
02732
02733
              struct combination_helper<T, 0, n> {
02734
                  using type = typename FractionField<T>::one;
02735
              };
02736
02737
              template<typename T, size_t k, size_t n>
02738
              struct combination {
02739
                  using type = typename internal::combination_helper<T, k, n>::type::x;
02740
                  static constexpr typename T::inner_type value
02741
                               internal::combination_helper<T, k, n>::type::template get<typename</pre>
      T::inner_type>();
02742
02743
          } // namespace internal
02744
02747
          template<typename T, size_t k, size_t n>
          using combination_t = typename internal::combination<T, k, n>::type;
02748
02749
          template<typename T, size_t k, size_t n>
inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02754
02755
02756
02757
          namespace internal {
02758
              template<typename T, size_t m>
02759
              struct bernoulli;
02760
              template<typename T, typename accum, size_t k, size_t m>
struct bernoulli_helper {
02761
02762
```

```
using type = typename bernoulli_helper<
02764
02765
                       addfractions_t<T,
02766
                           accum,
02767
                           {\tt mulfractions\_t < T,}
                               makefraction_t<T,
02768
02769
                                    combination_t<T, k, m + 1>,
02770
                                    typename T::one>,
02771
                                typename bernoulli<T, k>::type
02772
02773
                       >,
02774
                       k + 1,
02775
                       m>::type;
02776
02777
02778
              template<typename T, typename accum, size_t m>
02779
              struct bernoulli_helper<T, accum, m, m> {
02780
                  using type = accum;
02782
02783
02784
02785
              template<typename T, size_t m>
02786
               struct bernoulli {
02787
                   using type = typename FractionField<T>::template mul_t<</pre>
02788
                      typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02789
02790
                       typename T::template val<static_cast<typename T::inner_type>(-1)>,
02791
                       typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02792
02793
                   >;
02794
02795
                   template<typename floatType>
02796
                   static constexpr floatType value = type::template get<floatType>();
02797
              };
02798
02799
              template<typename T>
02800
              struct bernoulli<T, 0> {
02801
                   using type = typename FractionField<T>::one;
02802
02803
                   template<typename floatType>
                   static constexpr floatType value = type::template get<floatType>();
02804
02805
              };
02806
          } // namespace internal
02807
02811
          template<typename T, size_t n>
02812
          using bernoulli_t = typename internal::bernoulli<T, n>::type;
02813
          template<typename FloatType, typename T, size_t n >
inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02818
02819
02820
02821
           // bell numbers
02822
          namespace internal {
              template<typename T, size_t n, typename E = void>
02823
02824
              struct bell_helper;
02825
02826
              template <typename T, size_t n>
02827
              struct bell_helper<T, n, std::enable_if_t<(n > 1)» {
02828
                   template<typename accum, size_t i, size_t stop>
02829
                   struct sum_helper {
02830
                   private:
                      using left = typename T::template mul_t<</pre>
02831
                                    combination_t<T, i, n-1>,
typename bell_helper<T, i>::type>;
02832
02833
                       using new_accum = typename T::template add_t<accum, left>;
02834
                   public:
02835
02836
                       using type = typename sum_helper<new_accum, i+1, stop>::type;
                   };
02837
02838
                   template<typename accum, size_t stop>
02840
                   struct sum_helper<accum, stop, stop> {
02841
                       using type = accum;
02842
02843
02844
                   using type = typename sum helper<typename T::zero, 0, n>::type;
02845
              };
02846
02847
              template<typename T>
02848
               struct bell_helper<T, 0> {
02849
                  using type = typename T::one;
02850
02851
02852
               template<typename T>
02853
               struct bell_helper<T, 1> {
02854
                   using type = typename T::one;
02855
02856
             // namespace internal
```

```
02857
          template<typename T, size_t n>
02861
02862
          using bell_t = typename internal::bell_helper<T, n>::type;
02863
02867
          template<typename T, size_t n>
static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
02868
02869
02870
02871
              template<typename T, int k, typename E = void>
02872
              struct alternate {};
02873
02874
              template<typename T, int k>
02875
              struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
02876
                  using type = typename T::one;
02877
                  static constexpr typename T::inner_type value = type::template get<typename</pre>
     T::inner_type>();
02878
              };
02879
02880
              template<typename T, int k>
              struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
02881
02882
                  using type = typename T::template sub_t<typename T::zero, typename T::one>;
02883
                  static constexpr typename T::inner_type value = type::template get<typename</pre>
     T::inner_type>();
02884
          };
} // namespace internal
02885
02886
02889
          template<typename T, int k>
02890
          using alternate_t = typename internal::alternate<T, k>::type;
02891
02894
          template<typename T, size_t k>
02895
          inline constexpr typename T::inner type alternate v = internal::alternate<T, k>::value;
02896
02897
02898
              template<typename T, int n, int k, typename E = void>
02899
              struct stirling_1_helper {};
02900
02901
              template<typename T>
              struct stirling_1_helper<T, 0, 0> {
02902
02903
                 using type = typename T::one;
02904
02905
02906
              template<typename T, int n>
              struct stirling_1_helper<T, n, 0, std::enable_if_t<(n > 0)» {
02907
02908
                  using type = typename T::zero;
02909
02910
02911
              template<typename T, int n>
              struct stirling_1_helper<T, 0, n, std::enable_if_t<(n > 0)» {
02912
                 using type = typename T::zero;
02913
02914
02915
02916
              template<typename T, int n, int k>
02917
              struct stirling_1_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)» {
02918
                  using type = typename T::template sub_t<
02919
                                   typename stirling_1_helper<T, n-1, k-1>::type,
02920
                                   typename T::template mul_t<</pre>
02921
                                       typename T::template inject_constant_t<n-1>,
02922
                                       typename stirling_1_helper<T, n-1, k>::type
02923
02924
          } // namespace internal
02925
02926
02931
          template<typename T, int n, int k>
02932
          using stirling_1_signed_t = typename internal::stirling_1_helper<T, n, k>::type;
02933
02938
          template<typename T, int n, int k>
02939
          using stirling_1_unsigned_t = abs_t<typename internal::stirling_1_helper<T, n, k>::type>;
02940
02945
          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;
02946
02947
02952
          template<typename T, int n, int k>
02953
          static constexpr typename T::inner_type stirling_1_signed_v = stirling_1_signed_t<T, n, k>::v;
02954
02955
          namespace internal {
02956
              template<typename T, int n, int k, typename E = void>
02957
              struct stirling_2_helper {};
02958
              template<typename T, int n>
struct stirling_2_helper<T, n, n, std::enable_if_t<(n >= 0)» {
02959
02960
02961
                  using type = typename T::one;
02962
              };
02963
02964
              template<typename T, int n>
02965
              struct stirling_2_helper<T, n, 0, std::enable_if_t<(n > 0)» {
02966
                  using type = typename T::zero;
02967
              };
```

```
02969
               template<typename T, int n>
               struct stirling_2_helper<T, 0, n, std::enable_if_t<(n > 0)» {
02970
02971
                 using type = typename T::zero;
02972
02973
02974
               template<typename T, int n, int k>
02975
               struct stirling_2_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0) && (k < n) >> {
02976
                  using type = typename T::template add_t<
02977
                                     \label{typename} \mbox{typename stirling\_2\_helper<T, n-1, k-1>::type,}
02978
                                     typename T::template mul_t<</pre>
02979
                                         typename T::template inject constant t<k>,
02980
                                         typename stirling 2 helper<T, n-1, k>::type
02981
02982
02983
          } // namespace internal
02984
          template<typename T, int n, int k>
using stirling_2_t = typename internal::stirling_2_helper<T, n, k>::type;
02989
02990
02991
02996
          template<typename T, int n, int k>
02997
          static constexpr typename T::inner_type stirling_2_v = stirling_2_t<T, n, k>::v;
02998
02999
          namespace internal {
03000
               template<typename T>
03001
               struct pow_scalar {
03002
                   template<size_t p>
03003
                   static constexpr DEVICE INLINED T func (const T& x) { return p == 0 ? static_cast<T>(1) :
                       p % 2 == 0? func<p/2>(x) * func<p/2>(x) :
03004
                        x * func<p/2>(x) * func<p/2>(x);
03005
03006
                   }
03007
               };
03008
03009
               template<typename T, typename p, size_t n, typename E = void>
03010
               requires IsEuclideanDomain<T>
03011
               struct pow;
03012
03013
               template<typename T, typename p, size_t n>
               struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)» {
03014
03015
                  using type = typename T::template mul_t<
03016
                        typename pow<T, p, n/2>::type,
03017
                       typename pow<T, p, n/2>::type
03018
03019
               };
03020
               template<typename T, typename p, size_t n>
03021
03022
               struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)\times {
03023
                   using type = typename T::template mul_t<</pre>
03024
                       p,
03025
                        typename T::template mul t<
03026
                            typename pow<T, p, n/2>::type,
03027
                            typename pow<T, p, n/2>::type
03028
03029
                   >;
              };
03030
03031
03032
               template<typename T, typename p, size_t n>
03033
               struct pow<T, p, n, std::enable_if_t<n == 0» { using type = typename T::one; };</pre>
03034
          } // namespace internal
03035
03040
          template<typename T, typename p, size_t n>
03041
          using pow_t = typename internal::pow<T, p, n>::type;
03042
03047
          template<typename T, typename p, size_t n>
03048
          static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
03049
03050
          template<typename T, size_t p>
static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
03051
      internal::pow scalar<T>::template func(x); }
03052
03053
           namespace internal {
03054
               template<typename, template<typename, size_t> typename, class>
03055
               struct make_taylor_impl;
03056
               template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...» {</pre>
03057
03058
03059
                 using type = typename polynomial<FractionField<T»::template val<typename coeff_at<T,
      Is>::type...>;
03060
              };
03061
          }
03062
03067
          template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
03068
          using taylor = typename internal::make_taylor_impl<</pre>
03069
03070
               coeff_at,
03071
               internal::make index sequence reverse<deg + 1>>::type;
03072
```

```
namespace internal {
03074
              template<typename T, size_t i>
03075
               struct exp_coeff {
03076
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03077
03078
03079
               template<typename T, size_t i, typename E = void>
03080
               struct sin_coeff_helper {};
03081
03082
               template<typename T, size_t i>
               struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
03083
03084
03085
03086
03087
               template<typename T, size_t i>
               struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
    using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03088
03089
03090
03091
03092
               template<typename T, size_t i>
03093
               struct sin_coeff {
03094
                   using type = typename sin_coeff_helper<T, i>::type;
03095
03096
03097
               template<typename T, size_t i, typename E = void>
03098
               struct sh_coeff_helper {};
03099
               template<typename T, size_t i>
03100
03101
               struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0  {
                  using type = typename FractionField<T>::zero;
03102
03103
03104
03105
               template<typename T, size_t i>
03106
               03107
                   using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03108
03109
03110
               template<typename T, size_t i>
03111
               struct sh_coeff {
03112
                 using type = typename sh_coeff_helper<T, i>::type;
03113
0.3114
               template<typename T, size_t i, typename E = void>
03115
03116
               struct cos_coeff_helper {};
03117
03118
               template<typename T, size_t i>
               struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
    using type = typename FractionField<T>::zero;
03119
03120
03121
03122
03123
               template<typename T, size_t i>
03124
               struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03125
                  using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03126
03127
03128
               template<typename T, size t i>
              struct cos_coeff {
03130
                  using type = typename cos_coeff_helper<T, i>::type;
03131
03132
03133
              template<typename T, size t i, typename E = void>
03134
              struct cosh coeff helper {};
03135
03136
               template<typename T, size_t i>
03137
               struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03138
                  using type = typename FractionField<T>::zero;
03139
03140
03141
               template<typename T, size_t i>
03142
               struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03143
                 using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03144
0.3145
               template<typename T, size_t i>
03146
03147
               struct cosh coeff {
03148
                   using type = typename cosh_coeff_helper<T, i>::type;
03149
03150
03151
               template<typename T, size_t i>
               struct geom_coeff { using type = typename FractionField<T>::one; };
03152
03153
03154
03155
               template<typename T, size_t i, typename E = void>
03156
               struct atan_coeff_helper;
03157
              template<typename T, size_t i>
struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03158
03159
```

```
using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;;
03161
03162
03163
               template<typename T, size_t i>
               struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
03164
03165
03166
03167
03168
               template<typename T, size_t i>
03169
               struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
03170
03171
               template<typename T, size_t i, typename E = void>
03172
               struct asin coeff helper;
03173
03174
               template<typename T, size_t i>
03175
               struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1  {
03176
                   using type = makefraction_t<T,</pre>
                       factorial_t<T, i - 1>,
03177
                       typename T::template mul_t<
03178
                            typename T::template val<i>,
03179
03180
                            T::template mul_t<
03181
                                pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
                                pow<T, factorial_t<T, i / 2>, 2
03182
03183
03184
03185
                       »;
03186
               };
03187
03188
               template<typename T, size_t i>
               struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0\times {
03189
                   using type = typename FractionField<T>::zero;
03190
03191
03192
03193
               template<typename T, size_t i>
03194
               struct asin_coeff {
                   using type = typename asin_coeff_helper<T, i>::type;
03195
03196
               };
03197
03198
               template<typename T, size_t i>
03199
               struct lnp1_coeff {
03200
                   using type = makefraction_t<T,</pre>
                       alternate_t<T, i + 1>,
03201
                       typename T::template val<i>;
03202
03203
               };
03204
03205
               template<typename T>
03206
               struct lnpl_coeff<T, 0> { using type = typename FractionField<T>::zero; };
03207
               template<typename T, size_t i, typename E = void>
03208
03209
               struct asinh coeff helper:
03210
03211
               template<typename T, size_t i>
03212
               struct asinh\_coeff\_helper<T, i, std::enable\_if\_t<(i & 1) == 1> {
03213
                   using type = makefraction_t<T,</pre>
                       typename T::template mul_t<</pre>
03214
03215
                           alternate_t<T, i / 2>,
03216
                            factorial_t<T, i - 1>
03217
03218
                        typename T::template mul_t<
03219
                            typename T::template mul_t<</pre>
                                typename T::template val<i>,
pow_t<T, factorial_t<T, i / 2>, 2>
03220
03221
03222
03223
                            pow_t<T, typename T::template inject_constant_t<4>, i / 2>
03224
03225
                  >;
03226
               };
03227
03228
               template<typename T, size_t i>
03229
               struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03230
                  using type = typename FractionField<T>::zero;
03231
03232
               template<typename T, size_t i>
03233
03234
               struct asinh coeff {
03235
                   using type = typename asinh_coeff_helper<T, i>::type;
03236
03237
03238
               template<typename T, size_t i, typename E = void>
03239
               struct atanh coeff helper;
03240
03241
               template<typename T, size_t i>
03242
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03243
                   // 1/i
03244
                   using type = typename FractionField<T>:: template val<</pre>
03245
                       typename T::one,
03246
                       typename T::template inject_constant_t<i>;;
```

```
03247
              };
03248
03249
               template<typename T, size_t i>
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03250
                   using type = typename FractionField<T>::zero;
03251
03252
               };
03253
03254
               template<typename T, size_t i>
03255
              struct atanh_coeff {
03256
                   using type = typename atanh_coeff_helper<T, i>::type;
03257
03258
03259
               template<typename T, size_t i, typename E = void>
03260
              struct tan_coeff_helper;
03261
               template<typename T, size_t i>
03262
               struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
    using type = typename FractionField<T>::zero;
03263
03264
03265
03266
03267
               template<typename T, size_t i>
03268
               struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
03269
               private:
                   // 4^((i+1)/2)
03270
03271
                   using _4p = typename FractionField<T>::template inject_t<</pre>
03272
                       pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
03273
                   // 4^((i+1)/2)
03274
                   using _4pm1 = typename FractionField<T>::template
using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;</pre>
03276
03277
                   using dividend = typename FractionField<T>::template mul_t<
03278
                       altp,
03279
                       FractionField<T>::template mul_t<
03280
                       FractionField<T>::template mul t<
03281
03282
                       4pm1,
                       bernoulli_t<T, (i + 1)>
03283
03284
03285
03286
               public:
03287
                  using type = typename FractionField<T>::template div_t<dividend,</pre>
03288
03289
                       typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03290
              };
03291
03292
               template<typename T, size_t i>
03293
               struct tan coeff {
03294
                   using type = typename tan_coeff_helper<T, i>::type;
03295
03296
03297
               template<typename T, size_t i, typename E = void>
03298
               struct tanh_coeff_helper;
03299
03300
               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;
03301
03302
03303
03304
03305
               template<typename T, size_t i>
               struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
03306
03307
               private:
03308
                   using _4p = typename FractionField<T>::template inject_t<</pre>
                   pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
using _4pml = typename FractionField<T>::template
03309
03310
      sub_t<_4p, typename FractionField<T>::one>;
03311
                   using dividend =
03312
                       typename FractionField<T>::template mul t<
03313
                            4p.
03314
                            typename FractionField<T>::template mul_t<</pre>
03315
                               \_4pm1,
03316
                                bernoulli_t<T, (i + 1) >>::type;
              public:
03317
                using type = typename FractionField<T>::template div_t<dividend,</pre>
03318
                       FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03319
03320
              } ;
03321
03322
               template<typename T, size_t i>
03323
               struct tanh coeff {
                 using type = typename tanh_coeff_helper<T, i>::type;
03324
03325
03326
          } // namespace internal
03327
03331
          template<typename Integers, size_t deg>
03332
          using exp = taylor<Integers, internal::exp_coeff, deg>;
03333
03337
          template<typename Integers, size t deg>
```

```
using expml = typename polynomial<FractionField<Integers>>::template sub_t
03339
03340
                      typename polynomial<FractionField<Integers>>::one>;
03341
03345
                template<typename Integers, size_t deg>
               using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
03346
03347
03351
                template<typename Integers, size_t deg>
03352
                using atan = taylor<Integers, internal::atan_coeff, deg>;
03353
03357
                template<typename Integers, size_t deg>
03358
               using sin = taylor<Integers, internal::sin coeff, deg>;
03359
03363
                template<typename Integers, size_t deg>
03364
                using sinh = taylor<Integers, internal::sh_coeff, deg>;
03365
03370
                template<typename Integers, size_t deg>
03371
                using cosh = taylor<Integers, internal::cosh_coeff, deq>;
03372
03377
                template<typename Integers, size_t deg>
03378
                using cos = taylor<Integers, internal::cos_coeff, deg>;
03379
03384
                template<typename Integers, size_t deg>
               using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
03385
03386
03391
                template<typename Integers, size_t deg>
03392
                using asin = taylor<Integers, internal::asin_coeff, deg>;
03393
03398
                template<typename Integers, size_t deg>
03399
               using asinh = taylor<Integers, internal::asinh_coeff, deg>;
03400
03405
                template<typename Integers, size t deg>
03406
                using atanh = taylor<Integers, internal::atanh_coeff, deg>;
03407
03412
                template<typename Integers, size_t deg>
03413
                using tan = taylor<Integers, internal::tan_coeff, deg>;
03414
03419
                template<typename Integers, size_t deg>
03420
                using tanh = taylor<Integers, internal::tanh_coeff, deg>;
03421 }
             // namespace aerobus
03422
03423 // continued fractions
03424 namespace aerobus {
03427
               template<int64_t... values>
03428
               struct ContinuedFraction {};
03429
03432
               template<int64_t a0>
03433
               struct ContinuedFraction<a0> {
                      using type = typename q64::template inject_constant_t<a0>;
03435
                      static constexpr double val = static_cast<double>(a0);
03437
03438
               };
03439
03443
               template<int64_t a0, int64_t... rest>
03444
                struct ContinuedFraction<a0, rest...> {
03446
                      using type = q64::template add_t<
                                   typename q64::template inject_constant_t<a0>,
03447
                                   typename q64::template div_t<
03448
03449
                                         typename q64::one,
03450
                                         typename ContinuedFraction<rest...>::type
03451
03452
03454
                      static constexpr double val = type::template get<double>();
03455
               };
03456
03460
               using PI_fraction =
         ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1>;
03462
               using E_fraction =
         ContinuedFraction<2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1>;
              using SQRT2_fraction =
03464
         03466
              using SQRT3_fraction =
         ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 
          // NOLINT
03467 } // namespace aerobus
03468
03469 // known polynomials
03470 namespace aerobus {
03471
                // CChebyshev
03472
               namespace internal {
                      template<int kind, size_t deg, typename I>
03473
03474
                      struct chebyshev_helper {
03475
                            using type = typename polynomial<I>::template sub_t<
                                   typename polynomial<I>::template mul_t<
03476
03477
                                         typename polynomial<I>::template mul_t<
03478
                                               typename polynomial<I>::template inject_constant_t<2>,
03479
                                                typename polynomial<I>::X>
03480
                                         typename chebyshev_helper<kind, deg - 1, I>::type
```

```
03481
03482
                      typename chebyshev_helper<kind, deg - 2, I>::type
03483
03484
              };
03485
03486
              template<tvpename I>
03487
              struct chebyshev_helper<1, 0, I> {
03488
                  using type = typename polynomial<I>::one;
03489
03490
              template<tvpename I>
03491
03492
              struct chebyshev_helper<1, 1, I> {
03493
                  using type = typename polynomial<I>::X;
03494
03495
03496
              template<typename I>
              struct chebyshev_helper<2, 0, I> {
03497
03498
                 using type = typename polynomial<I>::one;
03499
03500
03501
              template<typename I>
03502
              struct chebyshev_helper<2, 1, I> {
                  using type = typename polynomial<I>::template mul_t<</pre>
03503
03504
                      typename polynomial<I>::template inject_constant_t<2>,
03505
                      typename polynomial<I>::X>;
03506
03507
          } // namespace internal
03508
          // Laguerre
03509
03510
          namespace internal {
              template<size_t deg, typename I>
03511
03512
              struct laguerre_helper
03513
                 using Q = FractionField<I>;
03514
                  using PQ = polynomial<Q>;
03515
               private:
03516
                  // Lk = (1 / k) * ((2 * k - 1 - x) * 1km1 - (k - 2)Lkm2)
03517
                  using lnm2 = typename laguerre_helper<deg - 2, I>::type;
03519
                  using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03520
                  // -x + 2k-1
03521
                  using p = typename PQ::template val<
                      \label{typename Q::template inject_constant_t<-1>,}
03522
                      typename Q::template inject_constant_t<2 * deg - 1»;</pre>
03523
03524
                  // 1/n
                  using factor = typename PQ::template inject_ring_t<</pre>
03525
03526
                      typename Q::template val<typename I::one, typename I::template
     inject_constant_t<deg>>;
03527
03528
               public:
                  using type = typename PQ::template mul_t <</pre>
03529
03530
                      factor,
03531
                       typename PQ::template sub_t<
03532
                          typename PQ::template mul_t<
03533
03534
                               lnm1
03535
03536
                          typename PQ::template mul_t<
03537
                               typename PQ::template inject_constant_t<deg-1>,
03538
03539
03540
03541
                  >;
03542
              };
03543
03544
              template<typename I>
03545
              struct laguerre_helper<0, I> {
03546
                  using type = typename polynomial<FractionField<I>::one;
03547
03548
              template<typename I>
03550
              struct laguerre_helper<1, I> {
               private:
03551
03552
                  using PQ = polynomial<FractionField<I»;
03553
               public:
03554
                  using type = typename PO::template sub t<typename PO::one, typename PO::X>;
03555
03556
          } // namespace internal
03557
          // Bernstein
03558
03559
          namespace internal {
03560
             template<size_t i, size_t m, typename I, typename E = void>
03561
              struct bernstein_helper {};
03562
03563
              template<typename I>
03564
              struct bernstein_helper<0, 0, I> {
                  using type = typename polynomial<I>::one;
03565
03566
              };
```

```
03568
              template<size_t i, size_t m, typename I>
              03569
03570
03571
               private:
03572
                 using P = polynomial<I>;
03573
              public:
03574
                 using type = typename P::template mul_t<
03575
                          typename P::template sub_t<typename P::one, typename P::X>,
03576
                          typename bernstein_helper<i, m-1, I>::type>;
03577
              };
03578
              03579
03580
03581
03582
               private:
                 using P = polynomial<I>;
03583
               public:
03584
03585
                 using type = typename P::template mul_t<
03586
                         typename P::X,
03587
                         typename bernstein_helper<i-1, m-1, I>::type>;
03588
03589
              template<size_t i, size_t m, typename I>
struct bernstein_helper<i, m, I, std::enable_if_t<</pre>
03590
03591
03592
                        (m > 0) && (i > 0) && (i < m)» {
03593
               private:
03594
                 using P = polynomial<I>;
03595
               public:
03596
                 using type = typename P::template add_t<
03597
                         typename P::template mul_t<
03598
                              typename P::template sub_t<typename P::one, typename P::X>,
03599
                              typename bernstein_helper<i, m-1, I>::type>,
03600
                          typename P::template mul_t<
03601
                             typename P::X,
                             typename bernstein_helper<i-1, m-1, I>::type»;
03602
03603
          } // namespace internal
03604
03605
03606
          // AllOne polynomials
03607
          namespace internal {
03608
             template<size_t deg, typename I>
              struct AllOneHelper {
03609
03610
                 using type = aerobus::add_t<
                     typename polynomial<I>::one,
03611
03612
                      typename aerobus::mul_t<
03613
                         typename polynomial<I>::X,
03614
                         typename AllOneHelper<deg-1, I>::type
03615
                     »;
03616
              };
03617
03618
              template<typename I>
03619
              struct AllOneHelper<0, I> {
03620
                using type = typename polynomial<I>::one;
03621
          } // namespace internal
03622
03623
          // Bessel polynomials
03624
03625
          namespace internal {
03626
             template<size_t deg, typename I>
              struct BesselHelper {
03627
03628
              private:
03629
                 using P = polynomial<I>;
03630
                  using factor = typename P::template monomial_t<
03631
                      typename I::template inject_constant_t<(2*deg - 1)>,
                     1>;
03632
03633
               public:
                 using type = typename P::template add_t<
03634
                      typename P::template mul_t<
03635
03636
                         factor,
03637
                         typename BesselHelper<deg-1, I>::type
03638
03639
                      typename BesselHelper<deg-2, I>::type
03640
                 >;
03641
              };
03642
03643
              template<typename I>
03644
              struct BesselHelper<0, I> {
03645
                 using type = typename polynomial<I>::one;
03646
03647
03648
              template<typename I>
03649
              struct BesselHelper<1, I> {
              private:
03650
03651
                 using P = polynomial<I>;
03652
               public:
03653
                 using type = typename P::template add_t<</pre>
```

```
typename P::one,
03655
                       typename P::X
03656
03657
               };
          } // namespace internal
03658
03659
03660
          namespace known_polynomials {
03662
               enum hermite_kind {
03664
                 probabilist,
03666
                   physicist
              };
03667
          }
03668
03669
03670
          // hermite
03671
          namespace internal {
03672
               template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03673
               struct hermite_helper {};
03674
03675
               template<size_t deg, typename I>
03676
               struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
03677
03678
                  using hnm1 = typename hermite_helper<deg - 1,
      known_polynomials::hermite_kind::probabilist, I>::type;
     using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::probabilist, I>::type;
03679
03680
03681
                public:
03682
                   using type = typename polynomial<I>::template sub_t<</pre>
03683
                       typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnml>,
                       typename polynomial<I>::template mul_t<
03684
03685
                           typename polynomial<I>::template inject_constant_t<deg - 1>,
03686
03687
03688
03689
              };
03690
               template<size_t deg, typename I>
struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03691
03692
03693
03694
                  using hnml = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
     I>::type;
03695
                   using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
     I>::tvpe;
03696
03697
                public:
03698
                   using type = typename polynomial<I>::template sub_t<</pre>
03699
                        // 2X Hn-1
03700
                       typename polynomial<I>::template mul_t<</pre>
03701
                           typename pi64::val<typename I::template inject_constant_t<2>,
typename I::zero>, hnml>,
03702
03703
03704
                       typename polynomial<I>::template mul_t<</pre>
03705
                           typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03706
                           hnm2
03707
03708
                   >;
03709
               } ;
03710
03711
               template<typename I>
03712
               struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03713
                   using type = typename polynomial<I>::one;
03714
              };
03716
               template<typename I>
03717
               struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03718
                   using type = typename polynomial<I>::X;
03719
03720
03721
               template<tvpename I>
03722
               struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03723
                  using type = typename pi64::one;
03724
03725
03726
               template<tvpename I>
03727
               struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03728
03729
                   using type = typename polynomial<I>::template val<</pre>
03730
                       typename I::template inject_constant_t<2>,
03731
                       typename I::zero>;
03732
               };
03733
          } // namespace internal
03734
03735
03736
          namespace internal {
03737
               template<size_t n, typename I>
03738
               struct legendre_helper {
03739
               private:
```

```
using Q = FractionField<I>;
03741
                  using PQ = polynomial<Q>;
03742
                  // 1/n constant
                  // (2n-1)/n X
03743
03744
                  using fact_left = typename PQ::template monomial_t<
03745
                      makefraction_t<I,
03746
                          typename I::template inject_constant_t<2*n-1>,
03747
                          typename I::template inject_constant_t<n>
03748
03749
                  1>;
                  // (n-1) / n
03750
03751
                  using fact_right = typename PQ::template val<
03752
                      makefraction_t<I,
03753
                          typename I::template inject_constant_t<n-1>,
03754
                          typename I::template inject_constant_t<n>>;
03755
03756
               public:
03757
                  using type = PQ::template sub_t<</pre>
03758
                          typename PQ::template mul_t<
03759
                               fact_left,
                               typename legendre_helper<n-1, I>::type
03760
03761
03762
                          typename PQ::template mul_t<
03763
                               fact_right,
03764
                               typename legendre_helper<n-2, I>::type
03765
03766
03767
03768
03769
              template<typename I>
03770
              struct legendre_helper<0, I> {
03771
                  using type = typename polynomial<FractionField<I>::one;
03772
03773
03774
              template<typename I>
              struct legendre_helper<1, I> {
03775
03776
                  using type = typename polynomial<FractionField<I>::X;
03777
03778
          } // namespace internal
03779
03780
          // bernoulli polynomials
03781
          namespace internal {
03782
              template<size t n>
              struct bernoulli_coeff {
03783
03784
                  template<typename T, size_t i>
03785
                  struct inner {
03786
                   private:
                      using F = FractionField<T>;
03787
                   public:
03788
03789
                      using type = typename F::template mul_t<
03790
                           typename F::template inject_ring_t<combination_t<T, i, n»,
03791
                          bernoulli_t<T, n-i>
03792
03793
                  };
03794
              };
03795
          } // namespace internal
03796
03797
          namespace internal {
03798
              template<size_t n>
03799
              struct touchard_coeff {
03800
                  template<typename T, size_t i>
03801
                  struct inner {
03802
                      using type = stirling_2_t<T, n, i>;
03803
03804
              } ;
03805
          } // namespace internal
03806
03807
          namespace internal {
03808
              template<typename I = aerobus::i64>
03809
              struct AbelHelper {
03810
              private:
03811
                  using P = aerobus::polynomial<I>;
03812
03813
               public:
                  // to keep recursion working, we need to operate on a\!*\!n and not just a
03814
03815
                  template<size_t deg, I::inner_type an>
03816
                  struct Inner {
03817
                      // abel(n, a) = (x-an) * abel(n-1, a)
03818
                      using type = typename aerobus::mul_t<
                          typename Inner<deg-1, an>::type,
03819
03820
                          typename aerobus::sub_t<typename P::X, typename P::template inject_constant_t<an>>
03821
                      >;
03822
03823
03824
                  // abel(0, a) = 1
03825
                  template<I::inner_type an>
03826
                  struct Inner<0, an> {
```

```
using type = P::one;
03828
03829
                                // abel(1, a) = X
03830
03831
                                template<I::inner_type an>
                                struct Inner<1, an> {
03832
                                      using type = P::X;
03834
03835
                 } // namespace internal
03836
03837
03839
                 namespace known polynomials {
03840
03849
                         template<size_t n, auto a, typename I = aerobus::i64>
03850
                         using abel = typename internal::AbelHelper<I>::template Inner<n, a*n>::type;
03851
                         template <size_t deg, typename I = aerobus::i64>
03859
03860
                         using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
03861
03871
                         template <size_t deg, typename I = aerobus::i64>
03872
                         using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
03873
03883
                         template <size_t deg, typename I = aerobus::i64>
                         using laguerre = typename internal::laguerre_helper<deg, I>::type;
03884
03885
03892
                         template <size_t deg, typename I = aerobus::i64>
                         using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
03893
         I>::type;
03894
03901
                         template <size_t deg, typename I = aerobus::i64>
03902
                         using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
03903
03914
                         template<size_t i, size_t m, typename I = aerobus::i64>
03915
                         using bernstein = typename internal::bernstein_helper<i, m, I>::type;
03916
                         template<size_t deg, typename I = aerobus::i64>
03926
03927
                         using legendre = typename internal::legendre_helper<deg, I>::type;
03928
03938
                         template<size_t deg, typename I = aerobus::i64>
03939
                         using bernoulli = taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
03940
03947
                         template<size_t deg, typename I = aerobus::i64>
03948
                         using allone = typename internal::AllOneHelper<deg, I>::type;
03949
03957
                         template<size_t deg, typename I = aerobus::i64>
03958
                         using bessel = typename internal::BesselHelper<deg, I>::type;
03959
                        template<size_t deg, typename I = aerobus::i64>
using touchard = taylor<I, internal::touchard_coeff<deg>::template inner, deg>;
03967
03968
03969
                       // namespace known_polynomials
03970 } // namespace aerobus
03971
03972
03973 #ifdef AEROBUS CONWAY IMPORTS
03974
03975 // conway polynomials
03976 namespace aerobus {
03980
                 template<int p, int n>
03981
                 struct ConwayPolynomial {};
03982
03983 #ifndef DO NOT DOCUMENT
03984
               #define ZPZV ZPZ::template val
03985
                 #define POLYV aerobus::polynomial<ZPZ>::template val
         template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
03987
                 template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
          POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
                 template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
03988
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1), ; // NOLINT template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
03989
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
                 template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
03990
         Template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus:.2pZ<2>, using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>; }; // NOLINT template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpZ<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>; using type = aerobus::zpZ<2>; using type =
03991
03992
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
03993
                 template<> struct ConwayPolynomial<2, 8> { using ZPZ = aerobus::zpz<2>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>; // NOLINT template<> struct ConwayPolynomial<2, 9> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>; }; //
03994
         template<> struct ConwayPolynomial<2, 10> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , Z
          ZPZV<1»; }; // NOLINT</pre>
          template<> struct ConwayPolynomial<2, 11> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
03996
```

```
ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                                       template<> struct ConwayPolynomial<2, 12> { using ZPZ = aerobus::zpz<2>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
03998
                                                        template<> struct ConwayPolynomial<2, 13> { using ZPZ = aerobus::zpz<2>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                                         template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
04000
                                  ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                                         template<> struct ConwayPolynomial<2, 16> { using ZPZ = aerobus::zpz<2>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT
  template<> struct ConwayPolynomial<2, 17> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                                         template<> struct ConwayPolynomial<2, 18> { using ZPZ = aerobus::zpz<2>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  template<> struct ConwayPolynomial<2, 19> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1</pre>
04004
                                                      template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type
 04005
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1</pre>; };
                                   // NOLINT
                                                      template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
04006
                                POLYV<ZPZV<1>, ZPZV<1»; };
                                                                                                                                                                                            // NOLINT
                                                       template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
                                 POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2»; }; // NOLINT
 04008
                                                        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 =
04009
                                 POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; }; // NOLINT
 04010
                                                        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
 04011
                                                     template<> struct ConwayPolynomial<3, 6> { using ZPZ = aerobus::zpz<3>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<3, 7> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>; }; // NOLINT
 04012
                                                        template<> struct ConwayPolynomial<3, 8> { using ZPZ = aerobus::zpz<3>; using type =
                                 POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>; };
 04014
                                                    template<> struct ConwayPolynomial<3, 9> { using ZPZ = aerobus::zpz<3>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; //
                                 NOLINT
                                                      template<> struct ConwayPolynomial<3, 10> { using ZPZ = aerobus::zpz<3>; using type
04015
                                  POLYY<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>
04016
                                                     template<> struct ConwayPolynomial<3, 11> { using ZPZ = aerobus::zpz<3>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04017
                                                        template<> struct ConwayPolynomial<3, 12> { using ZPZ = aerobus::zpz<3>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>,
                                  ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<3, 13> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                 template<> struct ConwayPolynomial<3, 14> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3 , Z
                                  ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<3, 15> { using ZPZ = aerobus::zpz<3>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<0
                                 ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; // NOLINT
    template<> struct ConwayPolynomial<3, 16> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04021
                                  ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
                                                        template<> struct ConwayPolynomial<3, 17> { using ZPZ = aerobus::zpz<3>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<1»; }; // NOLINT
   template<> struct ConwayPolynomial<3, 18> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04023
                                  ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<3, 19> { using ZPZ = aerobus::zpz<3>; using type =
 04024
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<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>,
04025
                                   ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2</pre>; };
                                   // NOLINT
 04026
                                                       template<> struct ConwayPolynomial<5, 1> { using ZPZ = aerobus::zpz<5>; using type =
                               POLYV<ZPZV<1>, ZPZV<3»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<5, 2> { using ZPZ = aerobus::zpz<5>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2»; };
                                                  template<> struct ConwayPolynomial<5, 3> { using ZPZ = aerobus::zpz<5>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
04029
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<5, 5> { using ZPZ = aerobus::zpz<5>; using type =
04030
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<3»; }; // NOLINT
04031
                                                  template<> struct ConwayPolynomial<5, 6> { using ZPZ = aerobus::zpz<5>; using type =
                             template<> struct ConwayPolynomial<5, 7> { using ZPZ = aerobus::zpz<5>; using type =
04032
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3»; }; // NOLINT
                                               template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
04033
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<5, 9> { using ZPZ = aerobus::zpz<5>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<3»; }; //
                                                template<> struct ConwayPolynomial<5, 10> { using ZPZ = aerobus::zpz<5>; using type =
04035
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<4>,
                              ZPZV<2»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<5, 11> { using ZPZ = aerobus::zpz<5>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<3>, ZPZV<3>; }; // NOLINT
                                               template<> struct ConwayPolynomial<5, 12> { using ZPZ = aerobus::zpz<5>; using type =
04037
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<2>; }; // NOLINT
                                                  template<> struct ConwayPolynomial<5, 13> { using ZPZ = aerobus::zpz<5>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
04039
                                                 template<> struct ConwayPolynomial<5, 14> { using ZPZ = aerobus::zpz<5>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<3>, ZPZV<3>, ZPZV<5>; J; // NOLINT template<> struct ConwayPolynomial<5, 15> { using ZPZ = aerobus::zpz<5>; using type =
04040
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>,
04041
                             ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<1>, 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»; }; // NOLINT</pre>
04043
                                               template<> struct ConwayPolynomial<5, 18> { using ZPZ = aerobus::zpz<5>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                               \texttt{ZPZV<2>, \ ZPZV<0>, \ ZPZV<2>, \ ZPZV<1>, \ ZPZV<0>, \ ZPZV<2>, \ ZPZV<2>, \ ZPZV<0>, \ ZPZV<2>, \ ZPZV<0>, \ ZPZV<0
                                                 template<> struct ConwayPolynomial<5, 19> { using ZPZ = aerobus::zpz<5>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<3»; };</pre>
                             template<> struct ConwayPolynomial<5, 20> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<3</pre>
                               // NOLINT
                                                 template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
                             POLYV<ZPZV<1>, ZPZV<4»; \}; // NOLINT
                                               template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
04047
                            POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<7, 3> { using ZPZ = aerobus::zpz<7>; using type =
04048
                             POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4»; }; // NOLINT
                                               template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type =
04049
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; }; // NOLINT
04050
                                                  template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>; }; // NOLINT template<> struct ConwayPolynomial<7, 6> { using ZPZ = aerobus::zpz<7>; using type =
04051
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<4>, ZPZV<6>, ZPZV<3»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<7, 7> { using ZPZ = aerobus::zpz<7>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<4»; };
04053
                                               template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<6>, ZPZV<2>, ZPZV<3»; }; // NOLINT
                                               template<> struct ConwayPolynomial<7, 9> { using ZPZ = aerobus::zpz<7>; using type
04054
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                             NOLINT
                                                 template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type =
04055
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<3>,
                               ZPZV<3»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<7, 11> { using ZPZ = aerobus::zpz<7>; using type =
04056
                             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<0>, ZPZV<2>, ZPZV<5>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<3>, ZPZV<3>, ZPZV<5>, ZPZV<5-, ZPZV<5
                                               template<> struct ConwayPolynomial<7, 13> { using ZPZ = aerobus::zpz<7>; using type =
04058
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<6>, ZPZV<0>, 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<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5</pre>
                             ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<3»; }; // NOLINT
  template<> struct ConwayPolynomial<7, 15> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04060
```

```
ZPZV<6>, ZPZV<6>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<4»; };</pre>
                                  template<> struct ConwayPolynomial<7, 16> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>; ZPZV<0>; ZPZV<0>, ZPZV<0 , Z
04062
                                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT</pre>
                                                            template<> struct ConwayPolynomial<7, 18> { using ZPZ = aerobus::zpz<7>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<6>, ZPZV<6 , ZPZV<6
                                   ZPZV<6>, ZPZV<5>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<2>, ZPZV<2>, ZPZV<3»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<7, 19> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
04064
                                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4»; }; //</pre>
04065
                                                       template<> struct ConwayPolynomial<7, 20> { using ZPZ = aerobus::zpz<7>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<6>, ZPZV<5>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3 , ZPZV<3
                                     // NOLINT
                                                           template<> struct ConwayPolynomial<11, 1> { using ZPZ = aerobus::zpz<11>; using type =
                                  POLYV<ZPZV<1>, ZPZV<9»; }; // NOLINT
                                                            template<> struct ConwayPolynomial<11, 2> { using ZPZ = aerobus::zpz<11>; using type =
                                  POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2»; }; // NOLINT
                                                         template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
 04068
                                POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, NOLINT template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type =
 04069
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
 04070
                                                         template<> struct ConwayPolynomial<11, 5> { using ZPZ = aerobus::zpz<11>; using type =
                                  template<> struct ConwayPolynomial<11, 6> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<6>, ZPZV<7>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<11, 7> { using ZPZ = aerobus::zpz<11>; using type =
 04071
 04072
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<9»; };
                                                           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
                                  template<> struct ConwayPolynomial<11, 9> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<8</pre>; }; //
 04074
                                  NOLINT
                                                           template<> struct ConwayPolynomial<11, 10> { using ZPZ = aerobus::zpz<11>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<10>, ZPZV<6>, ZPZV<6>,
                                     ZPZV<2»; }; // NOLINT</pre>
04076
                                                        template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<10>, ZPZV<9»; }; // NOLINT
   template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type
04077
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<5>, ZPZV<5>,
                                     ZPZV<6>, ZPZV<5>, ZPZV<2»; }; // NOLINT</pre>
04078
                                                        template<> struct ConwayPolynomial<11, 13> { using ZPZ = aerobus::zpz<11>; using type =
                                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                           template<> struct ConwayPolynomial<11, 14> { using ZPZ = aerobus::zpz<11>; using type
04079
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<0>, ZPZV<2>, ZPZV<5>, ZPZV<6>,
                                   ZPZV<4>, ZPZV<8>, ZPZV<6>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
04080
                                                        template<> struct ConwayPolynomial<11, 15> { using ZPZ = aerobus::zpz<11>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<1>, ZPZV<1
                                  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<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
04081
                                   ZPZV<1>, ZPZV<3>, ZPZV<5>, ZPZV<3>, ZPZV<10>, ZPZV<9>, ZPZV<2»; }; // NOLINT</pre>
                                                         template<> struct ConwayPolynomial<11, 17> { using ZPZ = aerobus::zpz<11>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<9»; ); // NOLINT template<> struct ConwayPolynomial<11, 18> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<8>, ZPZV<8>,
                                    \texttt{ZPZV<3>, ZPZV<9>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<9>, ZPZV<8>, ZPZV<2>, ZPZV<2>, ZPZV<2»; }; // \texttt{NOLINT} 
                                                       template<> struct ConwayPolynomial<11, 19> { using ZPZ = aerobus::zpz<11>; using type :
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<0>, ZPZV<0>
                                   NOLINT
                                  template<> struct ConwayPolynomial<11, 20> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                     ZPZV<9>, ZPZV<1>, ZPZV<5>, ZPZV<7>, ZPZV<2>, ZPZV<4>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<6</pre>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZP
                                     // NOLINT
04086
                                                           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 =
04087
                                  POLYV<ZPZV<1>, ZPZV<12>, ZPZV<2»; }; // NOLINT
                                                        template<> struct ConwayPolynomial<13, 3> { using ZPZ = aerobus::zpz<13>; using type =
 04088
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<13, 4> { using ZPZ = aerobus::zpz<13>; using type =
 04089
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<13, 5> { using ZPZ = aerobus::zpz<13>; using type =
 04090
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<11»; }; // NOLINT
                                                           template<> struct ConwayPolynomial<13, 6> { using ZPZ = aerobus::zpz<13>; using type =
                                 POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<11>, ZPZV<11>, ZPZV<2»; }; // NOLINT
                               template<> struct ConwayPolynomial<13, 7> { using ZPZ = aerobus::zpz<13>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<1>; // NOLINT
template<> struct ConwayPolynomial<13, 8> { using ZPZ = aerobus::zpz<13>; using type =
 04092
 04093
```

```
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
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<12>, ZPZV<12>, ZPZV<12>, ZPZV<12>, ZPZV<14>, ZPZV<14>, ZPZV<15, ZPZV<16, ZPZV<16, ZPZV<16, ZPZV<16, ZPZV<16, ZPZV<17, ZPZV<18, ZPZV<18,
                                           // NOLINT
                                                                     template<> struct ConwayPolynomial<13, 10> { using ZPZ = aerobus::zpz<13>; using type
04095
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>,
                                          ZPZV<2»; }; // NOLINT
                                                                      \texttt{template<>} \texttt{struct ConwayPolynomial<13, 11>} \{ \texttt{using ZPZ = aerobus::zpz<13>; using type aerobus
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<3>, ZPZV<11»; }; // 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<5>, ZPZV<8>, ZPZV<11>, ZPZV<3>, ZPZV<1>,
04097
                                          ZPZV<1>, ZPZV<4>, ZPZV<2»; }; // NOLINT</pre>
                                                                      template<> struct ConwayPolynomial<13, 13> { using ZPZ = aerobus::zpz<13>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<11»; }; // NOLINT
  template<> struct ConwayPolynomial<13, 14> { using ZPZ = aerobus::zpz<13>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV
                                          ZPZV<11>, ZPZV<7>, ZPZV<10>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
                                                                      template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>,
                                         ZPZV<2>, ZPZV<11>, ZPZV<10>, ZPZV<11>, ZPZV<8>, ZPZV<11»; }; // NOLINT
   template<> struct ConwayPolynomial<13, 16> { using ZPZ = aerobus::zpz<13>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
04101
                                         ZPZV89, ZPZV<2>, ZPZV<12>, ZPZV<12>, ZPZV<12>, ZPZV<4>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<12>, ZPZV<6>, ZPZV<12>, ZPZV<6>, ZPZV<12>, ZPZV<6>, ZPZV<12>, ZPZV<6>, ZPZV<12>, ZPZV<6>, ZPZV<6>, ZPZV<12>, 
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<6>, ZPZV<11»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<13, 18> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<11>, ZPZV<4>, ZPZV<11>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<9>, ZPZV<2»; }; // NOLINT
04104
                                                                       template<> struct ConwayPolynomial<13, 19> { using ZPZ = aerobus::zpz<13>; using type
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<0>, ZPZV<0</pre>
                                          NOLINT
                                         template<> struct ConwayPolynomial<13, 20> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<8>, ZPZV<8>, ZPZV<1>, ZPZV<2>; };
04105
                                                                      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 =
04107
                                         POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»: }: // NOLINT
                                                                     template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT
 04109
                                                                      template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
                                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<10>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<17, 5> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT
template<> struct ConwayPolynomial<17, 6> { using ZPZ = aerobus::zpz<17>; using type =
04110
 04111
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, 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<12>, ZPZV<14»; };
                                      template<> struct ConwayPolynomial<17, 8> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<0>, ZPZV<6>, ZPZV<3»; };
template<> struct ConwayPolynomial<17, 9> { using ZPZ = aerobus::zpz<17>; using type =
04113
 04114
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<8>, ZPZV<14»; };
                                         template<> struct ConwayPolynomial<17, 10> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<12>,
                                          ZPZV<3»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<17, 11> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                                                                                                                                                                                        // NOLINT
                                          ZPZV<5>, ZPZV<14»; };</pre>
                                                                 template<> struct ConwayPolynomial<17, 12> { using ZPZ = aerobus::zpz<17>; using type =
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<13>, ZPZV<6>,
                                          ZPZV<14>, ZPZV<9>, ZPZV<3»; }; // NOLINT</pre>
                                                                   template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type =
04118
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<14»; }; // NOLINT</pre>
                                                                      template<> struct ConwayPolynomial<17, 14> { using ZPZ = aerobus::zpz<17>; using type =
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
                                          ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::zpz<17>; using type
04120
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<16>, ZPZV<6>, ZPZV<14>, ZPZV<14»; }; // NOLINT
                                                                   template<> struct ConwayPolynomial<17, 16> { using ZPZ = aerobus::zpz<17>; using type
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<10>, Z
                                         ZPZV<5>, ZPZV<2>, ZPZV<12>, ZPZV<12>, ZPZV<12>, ZPZV<12>, 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<0>, ZPZV<0>,
04122
                                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<16>, ZPZV<16>, ZPZV<14»; }; // NOLINT
template<> struct ConwayPolynomial<17, 18> { using ZPZ = aerobus::zpz<17>; using type
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<5, ZPZV<1>, ZPZV<5, ZPZV<5
                                         ZPZV<7>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<11>, ZPZV<13>, ZPZV<13>, ZPZV<9>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<17, 19> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<114»; }; //</pre>
04125
                                       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<5>,
                          ZPZV<16>, ZPZV<14>, ZPZV<13>, ZPZV<3>, ZPZV<14>, ZPZV<9>, ZPZV<1>, ZPZV<13>, ZPZV<2>, ZPZV<5>,
                         ZPZV<3»; }; // NOLINT
                                          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 =
04128
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type =
04129
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2»; }; // NOLINT
04130
                                       template<> struct ConwayPolynomial<19, 5> { using ZPZ = aerobus::zpz<19>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<19, 6> { using ZPZ = aerobus::zpz<19>; using type =
04131
                        POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<2>; y; // 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<0>, ZPZV<6>, ZPZV<17»; };
                                          template<> struct ConwayPolynomial<19, 8> { using ZPZ = aerobus::zpz<19>, using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<10>, ZPZV<10>, ZPZV<3>, ZPZV<2»; };
04134
                                        template<> struct ConwayPolynomial<19, 9> { using ZPZ = aerobus::zpz<19>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<14>, ZPZV<16>, ZPZV<17»; };
                          // NOLINT
04135
                                          template<> struct ConwayPolynomial<19, 10> { using ZPZ = aerobus::zpz<19>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<13>, ZPZV<17>, ZPZV<3>, ZPZV<4+,
                          ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<19, 11> { using ZPZ = aerobus::zpz<19>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<17»; }; // NOLINT
                                          template<> struct ConwayPolynomial<19, 12> { using ZPZ = aerobus::zpz<19>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3
                        \label{template} $$\operatorname{template}<> \operatorname{struct\ ConwayPolynomial}<19,\ 13> \{ using\ ZPZ = aerobus:: zpz<19>;\ using\ type = POLYV<2PZV<1>,\ ZPZV<0>,\ ZPZV<0>,\
04138
                         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<11>,
                          ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<7>, ZPZV<2»; }; // NOLINT</pre>
04140
                                          template<> struct ConwayPolynomial<19, 15> { using ZPZ = aerobus::zpz<19>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>; // NOLINT template<> struct ConwayPolynomial<19, 16> { using ZPZ = aerobus::zpz<19>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>,
                          ZPZV<13>, ZPZV<0>, ZPZV<15>, ZPZV<9>, ZPZV<6>, ZPZV<14>, ZPZV<2»; }; // NOLINT</pre>
04142
                                        template<> struct ConwayPolynomial<19, 17> { using ZPZ = aerobus::zpz<19>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZV<0>; ZPZV<0>, ZPZV<0>; ZPZV<0>; ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZ
04143
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<9>, ZPZV<7>
                          ZPZV<17>, ZPZV<5>, ZPZV<0>, ZPZV<16>, ZPZV<5>, ZPZV<7>, ZPZV<3>, ZPZV<14>, ZPZV<2»; }; // NOLINT</pre>
04144
                                        template<> struct ConwayPolynomial<19, 19> { using ZPZ = aerobus::zpz<19>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<18>, ZPZV<17»; }; //</pre>
                         NOLINT
                                          template<> struct ConwayPolynomial<19, 20> { using ZPZ = aerobus::zpz<19>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<13>, ZPZV<0>, ZPZV<4>, ZPZV<7>, ZPZV<8>, ZPZV<6>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<6>, ZPZV<11>, ZPZV<2»;
                         }; // NOLINT
04146
                                         template<> struct ConwayPolynomial<23, 1> { using ZPZ = aerobus::zpz<23>; using type =
                        POLYV<ZPZV<1>, ZPZV<18»; }; // NOLINT
                                          template<> struct ConwayPolynomial<23, 2> { using ZPZ = aerobus::zpz<23>; using type =
                        POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; }; // NOLINT
04148
                                       template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; }; // NOLINT
                        template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type =
04149
04150
                        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 =
04151
                         \verb"POLYV<2PZV<1>, \verb"ZPZV<0>, \verb"ZPZV<1>, \verb"ZPZV<9>, \verb"ZPZV<9>, \verb"ZPZV<1>, \verb"ZPZV<5"; \verb"]; $ // NOLINT $ // NOLINT
                        template<> struct ConwayPolynomial<23, 7> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<18»; }; // NOLINT</pre>
04152
                                          template<> struct ConwayPolynomial<23, 8> { using ZPZ = aerobus::zpz<23>; using type =
04153
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5»; }; // NOLINT
                                        template<> struct ConwayPolynomial<23, 9> { using ZPZ = aerobus::zpz<23>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<8>, ZPZV<9>, ZPZV<18»; };
                         // NOLINT
04155
                                        template<> struct ConwayPolynomial<23, 10> { using ZPZ = aerobus::zpz<23>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6, ZPZV<
                         ZPZV<5»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<23, 11> { using ZPZ = aerobus::zpz<23>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<7>, ZPZV<18»; }; // NOLINT
                        template<> struct ConwayPolynomial<23, 12> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<21>, ZPZV<15>, ZPZV<14>, ZPZV<12>,
04157
```

```
ZPZV<18>, ZPZV<12>, ZPZV<5»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<23, 13> { using ZPZ = aerobus::zpz<23>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               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<1>, ZPZV<5>, ZPZV<1>, ZPZV<5>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<1>, ZPZV<5>, ZPZV<5-, ZPZV<5-,
04159
                                ZPZV<18>, ZPZV<19>, ZPZV<1>, ZPZV<22>, ZPZV<5»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<23, 15> { using ZPZ = aerobus::zpz<23>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
04161
                               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
                                 template<> struct ConwayPolynomial<23, 18> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<25, ZPZV<1>, ZPZV<16>, ZPZV<21>, ZPZV<11>, ZPZV<3>, ZPZV<5>; }; // NOLINT
                                                    template<> struct ConwayPolynomial<23, 19> { using ZPZ = aerobus::zpz<23>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5</pre>; };
                                NOLINT
                                                    template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type =
                               POLYV<ZPZV<1>, ZPZV<27»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
                               POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; }; // NOLINT
04167
                                                   template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<27»; }; // NOLINT template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
04168
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<25; }; // NOLINT template<> struct ConwayPolynomial<29, 5> { using ZPZ = aerobus::zpz<29>; using type =
04169
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<27»; }; // NOLINT
04170
                                                 template<> struct ConwayPolynomial<29, 6> { using ZPZ = aerobus::zpz<29>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<13>, ZPZV<23>, ZPZV<23>; ; // NOLINT template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>; ); // NOLINT template<> struct ConwayPolynomial<29, 8> { using ZPZ = aerobus::zpz<29>; using type =
04171
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<2*; };
04173
                                                 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
04174
                                                     template<> struct ConwayPolynomial<29, 10> { using ZPZ = aerobus::zpz<29>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<25>, ZPZV<8>, ZPZV<17>, ZPZV<17>, ZPZV<22>,
                                ZPZV<2»; }; // NOLINT</pre>
04175
                                                 template<> struct ConwayPolynomial<29, 11> { using ZPZ = aerobus::zpz<29>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<28>, ZPZV<27»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<29, 12> { using ZPZ = aerobus::zpz<29>; using type
04176
                               POLYY<ZPZV<1>, ZPZV<2>, ZPZV<2
04177
                                                  template<> struct ConwayPolynomial<29, 13> { using ZPZ = aerobus::zpz<29>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                     template<> struct ConwayPolynomial<29, 14> { using ZPZ = aerobus::zpz<29>; using type
04178
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<14>, ZPZV<10>,
                                ZPZV<21>, ZPZV<18>, ZPZV<27>, ZPZV<5>, ZPZV<2»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::zpz<29>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<13>, ZPZV<14>, ZPZV<6>, ZPZV<12>, ZPZV<12>, ZPZV<26>, ZPZV<27»; }; // NOLINT

template<> struct ConwayPolynomial<29, 16> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , 
                               ZPZV<2>, ZPZV<18>, ZPZV<23>, ZPZV<1>, ZPZV<27>, ZPZV<10>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<29, 17> { using ZPZ = aerobus::zpz<29>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<27»; }; // NOLINT
template<> struct ConwayPolynomial<29, 18> { using ZPZ = aerobus::zpz<29>; using type =
04182
                                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<6>, ZPZV<26>, ZPZV<2>, ZPZV<10>, ZPZV<8>, ZPZV<16>, ZPZV<19>, ZPZV<14>, ZPZV<2*; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<29, 19> { using ZPZ = aerobus::zpz<29>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>
                                NOLINT
04184
                                                    template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
                               POLYV<ZPZV<1>, ZPZV<28»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<31, 2> { using ZPZ = aerobus::zpz<31>; using type =
                               POLYV<ZPZV<1>, ZPZV<29>, ZPZV<3»; }; // NOLINT
04186
                                                   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 =
04187
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3»; };
                                                                                                                                                                                                                                                                                                                                      // NOLINT
                                                     template<> struct ConwayPolynomial<31, 5> { using ZPZ = aerobus::zpz<31>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; }; // NOLINT
                                                template<> struct ConwayPolynomial<31, 6> { using ZPZ = aerobus::zpz<31>; using type =
04189
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>>, ZPZV<16>, ZPZV<8>, ZPZV<8>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<28»; };
                                                          template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<12>, ZPZV<24>, ZPZV<3»; }; //
                                  NOLINT
 04192
                                  template<> struct ConwayPolynomial<31, 9> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV
                                                          template<> struct ConwayPolynomial<31, 10> { using ZPZ = aerobus::zpz<31>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<26>, ZPZV<13>, ZPZV<13>, ZPZV<13>,
                                   ZPZV<3»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<31, 11> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
 04194
                                                                                                                                                                            // NOLINT
                                   ZPZV<20>, ZPZV<28»; };</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<14>, ZPZV<28>, ZPZV<2>, ZPZV<9>,
                                    ZPZV<25>, ZPZV<12>, ZPZV<3»; }; // NOLINT</pre>
                                                          template<> struct ConwayPolynomial<31, 13> { using ZPZ = aerobus::zpz<31>; using type =
                                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                                                           template<> struct ConwayPolynomial<31, 14> { using ZPZ = aerobus::zpz<31>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
                                   \mbox{ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<6>, ZPZV<3»; }; // NOLINT
                                                        template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type =
 04198
                                  POLYV-ZPZV-1>, ZPZV-0>, ZPZV-0
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<24>, ZPZV<26>, ZPZV<28>, ZPZV<11>, ZPZV<19>, ZPZV<27>, ZPZV<27>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 17> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
 04200
                                                           template<> struct ConwayPolynomial<31, 18> { using ZPZ = aerobus::zpz<31>; using type
                                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<25>, ZPZV<24>,
                                   ZPZV<2>, ZPZV<7>, ZPZV<12>, ZPZV<11>, ZPZV<25>, ZPZV<25>, ZPZV<10>, ZPZV<6>, ZPZV<6»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
  04202
                                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7</pre>
  04203
                                                          template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
                                  POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
  04204
                                                       template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
                                  POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2»; }; // NOLINT
                                                          template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
  04205
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2»; }; // NOLINT
  04207
                                                       template<> struct ConwayPolynomial<37, 5> { using ZPZ = aerobus::zpz<37>; using type =
                                  \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<10>, \verb"ZPZV<35"; \verb"]; $ // \verb"NOLINT" | NOLINT" | NOLINT"
  04208
                                                          template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<4>, ZPZV<30>, ZPZV<2»; }; // NOLINT
  04209
                                                          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»; }; //
  04210
                                                        template<> struct ConwayPolynomial<37, 8> { using ZPZ = aerobus::zpz<37>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<20>, ZPZV<27>, ZPZV<27>, ZPZV<1>, ZPZV<2*, }; template<> struct ConwayPolynomial<37, 9> { using ZPZ = aerobus::zpz<37>; using type =
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   // NOLINT
  04211
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<20>, ZPZV<32>, ZPZV<35»; };
  04212
                                                        template<> struct ConwayPolynomial<37, 10> { using ZPZ = aerobus::zpz<37>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<11>, ZPZV<4>,
                                   ZPZV<2»; }; // NOLINT</pre>
                                                        template<> struct ConwayPolynomial<37, 11> { using ZPZ = aerobus::zpz<37>; using type =
04213
                                    \texttt{POLYV} < \texttt{PZV} < \texttt{1}>, \ \texttt{ZPZV} < \texttt{0}>, \ 
                                   ZPZV<2>, ZPZV<35»; }; // NOLINT</pre>
                                                           template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<10>, ZPZV<23̄>, ZPZV<23>,
                                   ZPZV<18>, ZPZV<33>, ZPZV<2»; }; // NOLINT</pre>
                                                           template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type
 04215
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35»; };</pre>
                                                                                                                                                                                                                                                                           // NOLINT
                                                            template<> struct ConwayPolynomial<37, 14> { using ZPZ = aerobus::zpz<37>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35>, ZPZV<35>, ZPZV<1>,
                                   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<1>, ZPZV<1>, ZPZV<1>, ZPZV<31>,
 04217
                                  ZPZV-28>, ZPZV-27>, ZPZV-13>, ZPZV-34>, ZPZV-33>, ZPZV-35»; }; // NOLINT
template<> struct ConwayPolynomial<37, 17> { using ZPZ = aerobus::zpz<37>; using type
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>; // NOLINT template<> struct ConwayPolynomial<37, 18> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<8>, ZPZV<8>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<20>, ZPZV<12>, ZPZV<12>, ZPZV<20>, ZPZV<12>, ZPZV<13>, ZPZV<20>, ZPZV<12>, ZPZV<14>, ZPZV<27>, ZPZV<20>, ZPZV<28; }; // NOLINT template<> struct ConwayPolynomial<37, 19> { using ZPZ = aerobus::zpz<37>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<23>, ZPZV<35»; };</pre>
                                                        template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type =
                                   POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
                         POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6»; }; // NOLINT
                                         template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<35»; }; // NOLINT
template<> struct ConwayPolynomial<41, 4> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<6»; }; // NOLINT
 04224
                                          template<> struct ConwayPolynomial<41, 5> { using ZPZ = aerobus::zpz<41>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<14>, ZPZV<35»; }; // NOLINT
                                        template<> struct ConwayPolynomial<41, 6> { using ZPZ = aerobus::zpz<41>; using type =
 04226
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<39>, ZPZV<6>, ZPZV<6>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<41, 7> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<35»; }; // NOLINT
04227
04228
                                          template<> struct ConwayPolynomial<41, 8> { using ZPZ = aerobus::zpz<41>; using type
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<32>, ZPZV<20>, ZPZV<6>, ZPZV<6»; };
                                       template<> struct ConwayPolynomial<41, 9> { using ZPZ = aerobus::zpz<41>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<5>, ZPZV<35»; };
                          // NOLINT
                         template<> struct ConwayPolynomial<41, 10> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<31>, ZPZV<8>, ZPZV<20>, ZPZV<30>,
04230
                          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<34>, ZPZV<34>, ZPZV<24>, ZPZV<21>, ZPZV<27>, ZPZV<6»; }; // NOLINT
04232
                                       template<> struct ConwayPolynomial<41, 13> { using ZPZ = aerobus::zpz<41>; using type =
04233
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<35»; }; // NOLINT
  template<> struct ConwayPolynomial<41, 14> { using ZPZ = aerobus::zpz<41>; using type =
04234
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>,
                          ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type =
04235
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35»; }; // NOLINT
    template<> struct ConwayPolynomial<41, 17> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04236
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35»; }; // NOLINT
                                           template<> struct ConwayPolynomial<41, 18> { using ZPZ = aerobus::zpz<41>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
                         ZPZV<23>, ZPZV<35>, ZPZV<38>, ZPZV<24>, ZPZV<12>, ZPZV<29>, ZPZV<10>, ZPZV<6>, ZPZV<6>, ZPZV<6>; }; // NOLINT
template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04238
                          ZPZV<0>, ZPZV<10>, Z
04239
                                          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 =
04240
                         POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3»; }; // NOLINT
                                          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 =
                         template<> struct ConwayPolynomial<43, 5> { using ZPZ = aerobus::zpz<43>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<40»; }; // NOLINT
template<> struct ConwayPolynomial<43, 6> { using ZPZ = aerobus::zpz<43>; using type =
 04243
04244
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<21>, ZPZV<3»; }; // NOLINT
                                        template<> struct ConwayPolynomial<43, 7> { using ZPZ = aerobus::zpz<43>; using type
 04245
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<42>, ZPZV<7>, ZPZV<40»; }; // NOLINT
 04246
                                          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<24>, ZPZV<39; }; //
                          NOLINT
                                          template<> struct ConwayPolynomial<43, 9> { using ZPZ = aerobus::zpz<43>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<40»; };
04248
                                        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
                                         template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<7>, ZPZV<40»; };</pre>
                                                                                                                        // NOLINT
04250
                                       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 =
04251
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<40»; };</pre>
                                                                                                                                                                                                   // NOLINT
04252
                                         template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<38>, ZPZV<22>, ZPZV<24>,
                         ZPZV<37>, ZPZV<18>, ZPZV<4>, ZPZV<49>, ZPZV<39>; Jr NOLINT
   template<> struct ConwayPolynomial<43, 15> { using ZPZ = aerobus::zpz<43>; using type :
04253
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>,
                         ZPZV<22>, ZPZV<42>, ZPZV<44>, ZPZV<15>, ZPZV<37>, ZPZV<40»; }; // NOLINT
template<> struct ConwayPolynomial<43, 17> { using ZPZ = aerobus::zpz<43>; using type =
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<40»; }; // NOLINT
template<> struct ConwayPolynomial<43, 18> { using ZPZ = aerobus::zpz<43>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<41>,
                          ZPZV<24>, ZPZV<7>, ZPZV<24>, ZPZV<29>, ZPZV<16>, ZPZV<34>, ZPZV<37>, ZPZV<18>, ZPZV<38*; }; // NOLINT
template<> struct ConwayPolynomial<43, 19> { using ZPZ = aerobus::zpz<43>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>, ZPZV<40»; }; //</pre>
                          NOLINT
                                            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 =
04259
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42»; }; // NOLINT
                                            template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type =
04260
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; }; // NOLINT
04261
                                         template<> struct ConwayPolynomial<47, 5> { using ZPZ = aerobus::zpz<47>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT template<> struct ConwayPolynomial<47, 6> { using ZPZ = aerobus::zpz<47>; using type =
04262
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<41>, ZPZV<45>; J; // NOLINT template<> struct ConwayPolynomial<47, 7> { using ZPZ = aerobus::zpz<47>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT
                                             template<> struct ConwayPolynomial<47, 8> { using ZPZ = aerobus::zpz<47>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5»; };
04265
                                          template<> struct ConwayPolynomial<47, 9> { using ZPZ = aerobus::zpz<47>; using type :
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<19>, ZPZV<19, ZPZV<1>, ZPZV<42»; };
                           // NOLINT
                                             template<> struct ConwayPolynomial<47, 10> { using ZPZ = aerobus::zpz<47>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42>, ZPZV<14>, ZPZV<18>, ZPZV<45>, ZPZV<45>,
                           ZPZV<5»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<47, 11> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04267
04268
                                             template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type =
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<46 , ZPZV<46 , ZPZV<46 , ZPZV<46 , ZPZV<47 , ZPZV<46 , ZPZV<47 , ZPZV
                          template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04269
                                             template<> struct ConwayPolynomial<47, 14> { using ZPZ = aerobus::zpz<47>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<30>, ZPZV<30>,
                          ZPZV<17>, ZPZV<24>, ZPZV<9>, ZPZV<32>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<47, 15> { using ZPZ = aerobus::zpz<47>; using type =
04271
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<3>, ZPZV<42»; }; // NOLINT template<> struct ConwayPolynomial<47, 17> { using ZPZ = aerobus::zpz<47>; using type =
04272
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<42»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<47, 18> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<41>, ZPZV<41>, ZPZV<41>, ZPZV<41>, ZPZV<41>, ZPZV<42>,
04273
                          ZPZV<26>, ZPZV<44>, ZPZV<22>, ZPZV<11>, ZPZV<5>, ZPZV<45>, ZPZV<33>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<47, 19> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 
04274
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<42»; }; //</pre>
04275
                                            template<> struct ConwayPolynomial<53, 1> { using ZPZ = aerobus::zpz<53>; using type =
                          POLYV<ZPZV<1>, ZPZV<51»; }; // NOLINT
                                            template<> struct ConwayPolynomial<53, 2> { using ZPZ = aerobus::zpz<53>; using type =
04276
                          POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<53, 3> { using ZPZ = aerobus::zpz<53>; using type =
04277
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
04278
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<38>, ZPZV<2»; }; // NOLINT

template<> struct ConwayPolynomial<53, 5> { using ZPZ = aerobus::zpz<53>; using type =
04279
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT
                                             template<> struct ConwayPolynomial<53, 6> { using ZPZ = aerobus::zpz<53>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<4>, ZPZV<45>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<53, 7> { using ZPZ = aerobus::zpz<53>; using type =
04281
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<51»; }; // NOLINT
04282
                                          template<> struct ConwayPolynomial<53, 8> { using ZPZ = aerobus::zpz<53>; using type =
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<1>, ZPZV<2»; };
                                             template<> struct ConwayPolynomial<53, 9> { using ZPZ = aerobus::zpz<53>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<5>, ZPZV<51»; };
                           // NOLINT
                          template<> struct ConwayPolynomial<53, 10> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<27>, ZPZV<27>, ZPZV<25, ZPZV<29>,
04284
                          ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<53, 11> { using ZPZ = aerobus::zpz<53>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<15>, ZPZV<51»; };</pre>
                                                                                                                               // NOLINT
                          template<> struct ConwayPolynomial<53, 12> { using ZPZ = aerobus::zpz<53>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<34>, ZPZV<4>, ZPZV<13>, ZPZV<10>, ZPZV<42>,
ZPZV<34>, ZPZV<41>, ZPZV<41>, ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<53, 13> { using ZPZ = aerobus::zpz<53>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<52>, ZPZV<28>, ZPZV<51»; }; // NOLINT</pre>
04288
                                          template<> struct ConwayPolynomial<53, 14> { using ZPZ = aerobus::zpz<53>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
                           ZPZV<0>, ZPZV<37>, ZPZV<12>, ZPZV<23>, ZPZV<2»; }; // NOLINT</pre>
```

```
template<> struct ConwayPolynomial<53, 15> { using ZPZ = aerobus::zpz<53>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<53, 17> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZV<0>,
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<51>, ZPZV<5
                                ZPZV<27>, ZPZV<0>, ZPZV<39>, ZPZV<44>, ZPZV<6>, ZPZV<8>, ZPZV<16>, ZPZV<15, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<53, 19> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5, ZPZV<0>, ZPZV<5, ZP
04292
                                                       template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
                                POLYV<ZPZV<1>, ZPZV<57»; };
                                                                                                                                                                                          // NOLINT
04294
                                                    template<> struct ConwayPolynomial<59, 2> { using ZPZ = aerobus::zpz<59>; using type =
                               POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
04295
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<2»; }; // NOLINT
04297
                                                      template<> struct ConwayPolynomial<59, 5> { using ZPZ = aerobus::zpz<59>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<57»; }; // NOLINT 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
04298
                                                       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<1>, ZPZV<57»; }; // NOLINT
04300
                                                   template<> struct ConwayPolynomial<59, 8> { using ZPZ = aerobus::zpz<59>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<32>, ZPZV<2>, ZPZV<50>, ZPZV<2»; };
                                 NOLINT
                                                    template<> struct ConwayPolynomial<59, 9> { using ZPZ = aerobus::zpz<59>; using type =
04301
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57»; };
                                                  template<> struct ConwayPolynomial<59, 10> { using ZPZ = aerobus::zpz<59>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28>, ZPZV<25>, ZPZV<4>, ZPZV<39>, ZPZV<15>,
                                 ZPZV<2»; }; // NOLINT</pre>
                                                      template<> struct ConwayPolynomial<59, 11> { using ZPZ = aerobus::zpz<59>; using type =
04303
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<6>, ZPZV<57»; };</pre>
                                                                                                                                                              // NOLINT
                                                    template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<21>, ZPZV<38>, ZPZV<8>, ZPZV<1>, ZPZV<2>; }; // NOLINT
                                                      template<> struct ConwayPolynomial<59, 13> { using ZPZ = aerobus::zpz<59>; using type
04305
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57»; };</pre>
                                                                                                                                                                                                                                                               // NOLINT
04306
                                                    template<> struct ConwayPolynomial<59, 14> { using ZPZ = aerobus::zpz<59>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<33>, ZPZV<51>, ZPZV<11>,
                                 04307
                                                    template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5, ZPZV<5>, ZPZV<5>, ZPZV<5, ZPZ
04308
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<57»; }; // NOLINT
template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59>; using type =
04309
                                POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<32>, ZPZ
                                                   template<> struct ConwayPolynomial<59, 19> { using ZPZ = aerobus::zpz<59>; using type
04310
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<57»; }; //</pre>
                                 NOLINT
04311
                                                     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 =
                                POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
04313
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; }; // NOLINT
template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2»; }; // NOLINT
04314
                                template<> struct ConwayPolynomial<61, 5> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<59»; }; // NOLINT
04315
04316
                                                   template<> struct ConwayPolynomial<61, 6> { using ZPZ = aerobus::zpz<61>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<3>, ZPZV<29>, ZPZV<29>, ZPZV<29; }; // NOLINT template<> struct ConwayPolynomial<61, 7> { using ZPZ = aerobus::zpz<61>; using type =
04317
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<59»; }; // NOLINT
                                                       template<> struct ConwayPolynomial<61, 8> { using ZPZ = aerobus::zpz<61>; using type =
                                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<1>, ZPZV<56>, ZPZV<2»; };
04319
                                                    template<> struct ConwayPolynomial<61, 9> { using ZPZ = aerobus::zpz<61>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<5
                                  // NOLINT
                                template<> struct ConwayPolynomial<61, 10> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<15>, ZPZV<44>, ZPZV<16>, ZPZV<6>,
04320
                                 ZPZV<2»; }; // NOLINT</pre>
04321
                                                   template<> struct ConwayPolynomial<61, 11> { using ZPZ = aerobus::zpz<61>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04322
                                                  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<1>, ZPZV<15>, ZPZV<2»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            /<0>, ZPZV<0>, ZPZV<3>, ZPZV<59»; }; // NOLINT
template<> struct ConwayPolynomial<61, 14> { using ZPZ = aerobus::zpz<61>; using type =
                           ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<59»; };</pre>
04324
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<48>, ZPZV<46>, ZPZV<11>,
                            ZPZV<8>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<2»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<61, 15> { using ZPZ = aerobus::zpz<61>; using type =
04325
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<35>, ZPZV<44>, ZPZV<25>, ZPZV<23>, ZPZV<51>, ZPZV<59»; }; // NOLINT
   template<> struct ConwayPolynomial<61, 17> { using ZPZ = aerobus::zpz<61>; using type =
04326
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<59»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<61, 18> { using ZPZ = aerobus::zpz<61>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<36>, ZPZV<13>,
                          ZPZV<36>, ZPZV<4>, ZPZV<32>, ZPZV<57>, ZPZV<42>, ZPZV<25>, ZPZV<25>, ZPZV<52>, ZPZV<52>, ZPZV<2»; }; // NOLINT
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>,
04328
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<59»; }; //</pre>
04329
                                            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 =
04330
                          POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
                                             template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65»; }; // NOLINT
                                            template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type =
04332
                          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 =
04333
04334
                          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 =
04335
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<67, 8> { using ZPZ = aerobus::zpz<67>; using type =
04336
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<46>, ZPZV<17>, ZPZV<64>, ZPZV<64>, ZPZV<5, //
                                             template<> struct ConwayPolynomial<67, 9> { using ZPZ = aerobus::zpz<67>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<49>, ZPZV<55>, ZPZV<65»; };
                           // NOLINT
                          \label{eq:convergence} template<> struct ConwayPolynomial<67, 10> \{ using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<0>, ZPZV<16>, ZPZV<7>, ZPZV<23>,
04338
                           ZPZV<2»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<67, 11> { using ZPZ = aerobus::zpz<67>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<9>, ZPZV<65»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<67, 12> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57>, ZPZV<27>, ZPZV<4>, ZPZV<55>, ZPZV<64>, ZPZV<21>, ZPZV<27>, ZPZV<22>, ZPZV<22>, ZPZV<23>, ZPZV<4>, ZPZV<64>, ZPZV<21>, ZPZV<27>, ZPZV<23>, ZPZV<21>, ZPZV<23>, ZPZV<23>, ZPZV<24>, ZPZV<25>, ZPZV<25>,
                                             template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type
04341
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<65»; };</pre>
                                                                                                                                                                                                                 // 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<2>, ZPZV<2>, ZPZV<2>, ZPZV<5>,
04342
                          ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>,
                            ZPZV<52>, ZPZV<41>, ZPZV<20>, ZPZV<21>, ZPZV<46>, ZPZV<65»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<67, 17> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04344
                                            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<0>, ZPZV<5>, ZPZV<63>, ZPZV<63>, ZPZV<52>, ZPZV<18>, ZPZV<55>, ZPZV<28>, ZPZV<28>, ZPZV<25>, ZPZV<55>, ZPZV<28>, ZPZV<29>, ZPZV<55>, ZPZV<55>, ZPZV<28>, ZPZV<28>, ZPZV<55>, ZPZV<55>, ZPZV<55>, ZPZV<28>, ZPZV<55>, ZPZV<55>
                          template<> struct ConwayPolynomial<67, 19> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04346
                           NOLINT
04347
                                              template<> struct ConwayPolynomial<71, 1> { using ZPZ = aerobus::zpz<71>; using type =
                          POLYV<ZPZV<1>, ZPZV<64»; }; // NOLINT
04348
                                           template<> struct ConwayPolynomial<71, 2> { using ZPZ = aerobus::zpz<71>; using type =
                          POLYV<ZPZV<1>, ZPZV<69>, ZPZV<77; }; // NOLINT template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
04349
                          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
04351
                                             template<> struct ConwayPolynomial<71, 5> { using ZPZ = aerobus::zpz<71>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<64»; }; // NOLINT
                                            template<> struct ConwayPolynomial<71, 6> { using ZPZ = aerobus::zpz<71>; using type =
04352
                          POLYV<2PZV<1>, ZPZV<0>, ZPŽV<1>, ZPZV<10>, ZPZV<13>, ZPZV<29>, ZPZV<7»; }; // NOLINT
04353
                                            template<> struct ConwayPolynomial<71, 7> { using ZPZ = aerobus::zpz<71>; using type
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<64»; }; // NoL template<> struct ConwayPolynomial<71, 8> { using ZPZ = aerobus::zpz<71>; using type
04354
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<22>, ZPZV<19>, ZPZV<7»; };
04355
                                            template<> struct ConwayPolynomial<71, 9> { using ZPZ = aerobus::zpz<71>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43>, ZPZV<62>, ZPZV<64>; };
04356
                                    template<> struct ConwayPolynomial<71, 10> { using ZPZ = aerobus::zpz<71>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<53>, ZPZV<17>, ZPZV<26>, ZPZV<40>, ZPZV<40>,
                       ZPZV<7»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<71, 11> { using ZPZ = aerobus::zpz<71>; using type =
04357
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<48>, ZPZV<64»; };</pre>
                                                                                                                     // NOLINT
                                     template<> struct ConwayPolynomial<71, 12> { using ZPZ = aerobus::zpz<71>; using type
04358
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28>, ZPZV<29>, ZPZV<25>, ZPZV<21>,
                       ZPZV<58>, ZPZV<23>, ZPZV<7»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<71, 13> { using ZPZ = aerobus::zpz<71>; using type =
04359
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    template<> struct ConwayPolynomial<71, 15> { using ZPZ = aerobus::zpz<71>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04361
                       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<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<64»; }; //</pre>
                       NOLINT
04363
                                       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 =
04364
                       POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5»; }; // NOLINT
04365
                                       template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68»; }; // NOLINT template<> struct ConwayPolynomial<73, 4> { using ZPZ = aerobus::zpz<73>; using type =
04366
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5»; };
                                                                                                                                                                                                                                                      // NOLINT
                                       template<> struct ConwayPolynomial<73, 5> { using ZPZ = aerobus::zpz<73>; using type =
04367
                       04368
                                       template<> struct ConwayPolynomial<73, 6> { using ZPZ = aerobus::zpz<73>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<45>, ZPZV<23>, ZPZV<48>, ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<73, 7> { using ZPZ = aerobus::zpz<73>; using type
04369
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<68»; }; // NOLINT
                                       template<> struct ConwayPolynomial<73, 8> { using ZPZ = aerobus::zpz<73>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<5>, 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<72>, ZPZV<15>, ZPZV<68»; };</pre>
04371
                       // NOLINT
                       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<33>, ZPZV<32>, ZPZV<69>,
                       ZPZV<5»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<73, 11> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<68»; }; // NOLINT
                                       template<> struct ConwayPolynomial<73, 12> { using ZPZ = aerobus::zpz<73>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<69>, ZPZV<52>, ZPZV<26>, ZPZV<26>, ZPZV<46>, ZPZV<46>, ZPZV<29>, ZPZV<25>, ZPZV<5>; }; // NOLINT
04375
                                      template<> struct ConwayPolynomial<73, 13> { using ZPZ = aerobus::zpz<73>; using type =
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        template<> struct ConwayPolynomial<73, 15> { using ZPZ = aerobus::zpz<73>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<10>, ZPZV<33>, ZPZV<57>, ZPZV<57>, ZPZV<62>, ZPZV<68»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<73, 17> { using ZPZ = aerobus::zpz<73>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
                                       template<> struct ConwayPolynomial<73, 19> { using ZPZ = aerobus::zpz<73>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68»; }; //</pre>
                       NOLINT
04379
                                       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 =
04380
                       POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3»; }; // NOLINT
                                       template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
04381
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76»; }; // NOLINT template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
04382
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<79, 5> { using ZPZ = aerobus::zpz<79>; using type =
04383
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<76»; }; // NOLINT
                                      template<> struct ConwayPolynomial<79, 6> { using ZPZ = aerobus::zpz<79>; using type =
04384
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<68>, ZPZV<3»; }; // NOLINT
04385
                                      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<4>, ZPZV<76»; }; // NOLINT
                                      template<> struct ConwayPolynomial<79, 8> { using ZPZ = aerobus::zpz<79>; using type =
04386
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<60>, ZPZV<59>, ZPZV<48>, ZPZV<3»; }; //
04387
                                      template<> struct ConwayPolynomial<79, 9> { using ZPZ = aerobus::zpz<79>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<57>, ZPZV<19>, ZPZV<76»; };
                       // NOLINT
04388
                                    template<> struct ConwavPolvnomial<79, 10> { using ZPZ = aerobus::zpz<79>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<51>, ZPZV<1>, ZPZV<30>, ZPZV<42>,
                          ZPZV<3»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<79, 11> { using ZPZ = aerobus::zpz<79>; using type =
                          \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                         ZPZV<3>, ZPZV<76»; }; // NOLINT
template<> struct ConwayPolynomial<79, 12> { using ZPZ = aerobus::zpz<79>; using type =
04390
                         POLYY<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<59>, ZPZV<59>, ZPZV<50>, ZPZ
                                         template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type =
04391
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type
04392
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<76»; }; // NOLINT</pre>
04393
                                        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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<76»; }; //
                          NOLINT
                                           template<> struct ConwayPolynomial<83, 1> { using ZPZ = aerobus::zpz<83>; using type =
                         POLYV<ZPZV<1>, ZPZV<81»; }; // NOLINT
                                            template<> struct ConwayPolynomial<83, 2> { using ZPZ = aerobus::zpz<83>; using type =
                         POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
04396
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT template<> struct ConwayPolynomial<83, 4> { using ZPZ = aerobus::zpz<83>; using type =
04397
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<83, 5> { using ZPZ = aerobus::zpz<83>; using type =
04398
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<81»; }; // NOLINT
04399
                                           template<> struct ConwayPolynomial<83, 6> { using ZPZ = aerobus::zpz<83>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<76>, ZPZV<32>, ZPZV<17>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<83, 7> { using ZPZ = aerobus::zpz<83>; using type =
04400
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<81»; };
                                           template<> struct ConwayPolynomial<83, 8> { using ZPZ = aerobus::zpz<83>; using type
04401
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<23>, ZPZV<42>, ZPZV<2»; }; //
                          NOLINT
                                          template<> struct ConwayPolynomial<83, 9> { using ZPZ = aerobus::zpz<83>; using type =
04402
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<24>, ZPZV<24>, ZPZV<181»; };
                                           template<> struct ConwayPolynomial<83, 10> { using ZPZ = aerobus::zpz<83>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5, ZPZV<5,
                          ZPZV<2»; }; // NOLINT</pre>
                         \label{template} $$\operatorname{template}<> \operatorname{struct\ ConwayPolynomial}<83,\ 11> \{ using\ ZPZ = aerobus:: zpz<83>;\ using\ type = POLYV<2PZV<1>,\ ZPZV<0>,\ ZPZV<0>,\
04404
                                                                                                                                // NOLINT
                          ZPZV<17>, ZPZV<81»; };</pre>
                         template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<12>, ZPZV<13>, ZPZV<31>, ZPZV<19>, ZPZV<65>,
                          ZPZV<55>, ZPZV<75>, ZPZV<2»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                            template<> struct ConwayPolynomial<83, 17> { using ZPZ = aerobus::zpz<83>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<81»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04408
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<81»; }; //</pre>
                                          template<> struct ConwayPolynomial<89, 1> { using ZPZ = aerobus::zpz<89>; using type =
                         POLYV<ZPZV<1>, ZPZV<86»; }; // NOLINT
                                            template<> struct ConwayPolynomial<89, 2> { using ZPZ = aerobus::zpz<89>; using type =
04410
                        POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; }; // NOLINT

template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
04411
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<86»; }; // NOLINT
                                            template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; }; // NOLINT
                                         template<> struct ConwayPolynomial<89, 5> { using ZPZ = aerobus::zpz<89>; using type =
04413
                         template<> struct ConwayPolynomial<89, 6> { using ZPZ = aerobus::zpz<89>; using type =
04414
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<80>, ZPZV<80>, ZPZV<15>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type
04415
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<86»; }; // NOLINT
04416
                                        template<> struct ConwayPolynomial<89, 8> { using ZPZ = aerobus::zpz<89>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<40>, ZPZV<79>, ZPZV<3»; }; //
                          NOLINT
                                         template<> struct ConwayPolynomial<89, 9> { using ZPZ = aerobus::zpz<89>; using type
04417
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<12>, ZPZV<6>, ZPZV<86»; };
                          // NOLINT
04418
                                          template<> struct ConwayPolynomial<89, 10> { using ZPZ = aerobus::zpz<89>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<33>, ZPZV<82>, ZPZV<52>, ZPZV<4>,
                          ZPZV<3»: 1: // NOLINT
                                            template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type
04419
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<26>, ZPZV<86»; }; // NOLINT</pre>
04420
                                           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
04421
                                         template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<86»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<89, 17> { using ZPZ = aerobus::zpz<89>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<86»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<89, 19> { using ZPZ = aerobus::zpz<89>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<34>, ZPZV<36»; };</pre>
                                                    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 =
                               POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<97, 3> { using ZPZ = aerobus::zpz<97>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<92»; };
                                                                                                                                                                                                                                                                                  // NOLINT
                                                     template<> struct ConwayPolynomial<97, 4> { using ZPZ = aerobus::zpz<97>; using type =
 04427
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<97, 5> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<92»; }; // NOLINT
04428
                                                    template<> struct ConwayPolynomial<97, 6> { using ZPZ = aerobus::zpz<97>; using type =
                               POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<5»; }; // NOLINT template<> struct ConwayPolynomial<97, 7> { using ZPZ = aerobus::zpz<97>; using type =
 04430
                              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<92»; }; // NOLINT template<> struct ConwayPolynomial<97, 8> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<5>, ZPZV<32>, ZPZV<32>, ZPZV<5»; };
04431
                                                    template<> struct ConwayPolynomial<97, 9> { using ZPZ = aerobus::zpz<97>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
                                 // NOLINT
                               template<> struct ConwayPolynomial<97, 10> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<66>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<20>,
04433
                                ZPZV<5»; }; // NOLINT</pre>
04434
                                                     template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<5>, ZPZV<92»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<97, 12> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<59>, ZPZV<81>, ZPZV<86>, ZPZV<86>,
04435
                                ZPZV<78>, ZPZV<94>, ZPZV<5»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<97, 13> { using ZPZ = aerobus::zpz<97>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<92»; };</pre>
                                                                                                                                                                                                                                                  // NOLINT
04437
                                                   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<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<15>, ZPZV<15>, ZPZV<15</pre>
04439
                                                    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 =
04440
                               POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<99»; }; // NOLINT
template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<78>, ZPZV<2»; }; // NOLINT
04442
                                                     template<> struct ConwayPolynomial<101, 5> { using ZPZ = aerobus::zpz<101>; using type =
04443
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<99»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<101, 6> { using ZPZ = aerobus::zpz<101>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<90>, ZPZV<20>, ZPZV<67>, ZPZV<2»; }; // NOLINT
04445
                                                     template<> struct ConwayPolynomial<101, 7> { using ZPZ = aerobus::zpz<101>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6101>; using type =
04446
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76>, ZPZV<29>, ZPZV<24>, ZPZV<2*; };
                                                  template<> struct ConwayPolynomial<101, 9> { using ZPZ = aerobus::zpz<101>; using type =
04447
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64>, ZPZV<47>, ZPZV<99»; };
                                 // NOLINT
04448
                                                    template<> struct ConwavPolynomial<101, 10> { using ZPZ = aerobus::zpz<101>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>>, ZPZV<40>, ZPZV<100>, ZPZV<52>,
                                ZPZV<2»; }; // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<101, 11> { using ZPZ = aerobus::zpz<101>; using type :
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<31>, ZPZV<99»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<101, 12> { using ZPZ = aerobus::zpz<101>; using type :
04450
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<79>, ZPZV<64>, ZPZV<39>, ZPZV<38>, ZPZV<48>, ZPZV<84>, ZPZV<21>, ZPZV<22>; }; // NOLINT
                                                   template<> struct ConwayPolynomial<101,
                                                                                                                                                                                                                                                                  13> { using ZPZ = aerobus::zpz<101>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                  template<> struct ConwayPolynomial<101, 17> { using ZPZ = aerobus::zpz<101>; using type =
04452
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<99»; };</pre>
                                                   template<> struct ConwayPolynomial<101, 19> { using ZPZ = aerobus::zpz<101>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                NOLINT
                                                   template<> struct ConwayPolynomial<103, 1> { using ZPZ = aerobus::zpz<103>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<98»; };
                                   template<> struct ConwayPolynomial<103, 2> { using ZPZ = aerobus::zpz<103>; using type =
                    POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<103, 3> { using ZPZ = aerobus::zpz<103>; using type =
 04456
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<98»; }; // NOLINT
                                   template<> struct ConwayPolynomial<103, 4> { using ZPZ = aerobus::zpz<103>; using type =
04457
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<88>, ZPZV<5»; }; // NOLINT
                                   template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
 04458
                    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<96>, ZPZV<30>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<103, 7> { using ZPZ = aerobus::zpz<103>; using type =
 04459
04460
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<98»; };
                                   template<> struct ConwayPolynomial<103, 8> { using ZPZ = aerobus::zpz<103>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<70>, ZPZV<71>, ZPZV<49>, ZPZV<5»; };
                                 template<> struct ConwayPolynomial<103, 9> { using ZPZ = aerobus::zpz<103>; using type =
04462
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<97>, ZPZV<97>, ZPZV<51>, ZPZV<98»; };
                                   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>
04464
                                  template<> struct ConwayPolynomial<103, 11> { using ZPZ = aerobus::zpz<103>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   template<> struct ConwayPolynomial<103, 12> { using ZPZ = aerobus::zpz<103>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<74>, ZPZV<23>, ZPZV<94>, ZPZV<94>, ZPZV<81>,
                      ZPZV<29>, ZPZV<88>, ZPZV<5»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<103, 13> { using ZPZ = aerobus::zpz<103>; using type
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
04467
                                   template<> struct ConwayPolynomial<103, 17> { using ZPZ = aerobus::zpz<103>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     // NOLINT
                    template<> struct ConwayPolynomial<103, 19> { using ZPZ = aerobus::zpz<103>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
04468
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<28»; }; //</pre>
                                   template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
                    POLYV<ZPZV<1>, ZPZV<105»; }; // NOLINT
04470
                                 template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
                    POLYV<ZPZV<1>, ZPZV<103>, ZPZV<2»; }; // NOLINT
                                   template<> struct ConwayPolynomial<107, 3> { using ZPZ = aerobus::zpz<107>; using type =
 04471
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<105»; }; // NOLINT
                                   template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; }; // NOLINT
 04473
                                 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 =
04474
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<52>, ZPZV<22>, ZPZV<79>, ZPZV<2»; }; // NOLINT
                                   template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<105»; }; //
                                 template<> struct ConwayPolynomial<107, 8> { using ZPZ = aerobus::zpz<107>; using type
 04476
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<24>, ZPZV<95>, ZPZV<2»; };
                    NOLINT
04477
                                   template<> struct ConwayPolynomial<107, 9> { using ZPZ = aerobus::zpz<107>; using type
                    POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<36>, ZPZV<105»; };
                                  template<> struct ConwayPolynomial<107, 10> { using ZPZ = aerobus::zpz<107>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<94>, ZPZV<61>, ZPZV<83>, ZPZV<83>, ZPZV<95>,
                     ZPZV<2»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<107, 11> { using ZPZ = aerobus::zpz<107>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                   // NOLINT
                     ZPZV<8>, ZPZV<105»; };</pre>
                                template<> struct ConwayPolynomial<107, 12> { using ZPZ = aerobus::zpz<107>; using type =
04480
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<61>, ZPZV<42>, ZPZV<57>, ZPZV<57>, ZPZV<29; }; // NOLINT
                                 template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type =
04481
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105»; };</pre>
                                                                                                                                                                     // NOLINT
                                   template<> struct ConwayPolynomial<107, 17> { using ZPZ = aerobus::zpz<107>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                              ZPZV<0>,
                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105»; }; // NOLINT
template<> struct ConwayPolynomial<107, 19> { using ZPZ = aerobus::zpz<107>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04483
                      ZPZV<0>, ZPZV<2+, ZPZV<2+, ZPZV<10*, }; //</pre>
                                   template<> struct ConwayPolynomial<109, 1> { using ZPZ = aerobus::zpz<109>; using type =
                    POLYV<ZPZV<1>, ZPZV<103»; }; // NOLINT
                                   template<> struct ConwayPolynomial<109, 2> { using ZPZ = aerobus::zpz<109>; using type =
04485
                    POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; }; // NOLINT
                                   template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<103»; }; // NOLINT template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
 04487
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<109, 5> { using ZPZ = aerobus::zpz<109>; using type =
 04488
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103»; }; // NOLINT
```

```
04489
                                                  template<> struct ConwayPolynomial<109, 6> { using ZPZ = aerobus::zpz<109>; using type =
                            POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<102>, ZPZV<6>, ZPZV<6>; ); // NOLINT template<> struct ConwayPolynomial<109, 7> { using ZPZ = aerobus::zpz<109>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<103»; }; // NOLINT template<> struct ConwayPolynomial<109, 8> { using ZPZ = aerobus::zpz<109>; using type =
 04491
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<34>, ZPZV<86>, ZPZV<6»; }; //
                             NOLINT
                                                  template<> struct ConwayPolynomial<109, 9> { using ZPZ = aerobus::zpz<109>; using type
 04492
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<93>, ZPZV<87>, ZPZV<103»; };
                               // NOLINT
04493
                                               template<> struct ConwayPolynomial<109, 10> { using ZPZ = aerobus::zpz<109>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<71>, ZPZV<55>, ZPZV<66>, ZPZV<69>,
                              ZPZV<6»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<109, 11> { using ZPZ = aerobus::zpz<109>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<11>, ZPZV<103»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<109, 12> { using ZPZ = aerobus::zpz<109>; using type =
                             POLYY<ZPZV<1>, ZPZV<2>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6 , ZPZV<6
                                                 template<> struct ConwayPolynomial<109, 13> { using ZPZ = aerobus::zpz<109>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<103»; };</pre>
                                                                                                                                                                                                                                       // NOLINT
                                               template<> struct ConwayPolynomial<109, 17> { using ZPZ = aerobus::zpz<109>; using type =
04497
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<10>,
                                                                                                                                                                                                                                                                                                                                                                                                                      // NOLINT
                                                  template<> struct ConwayPolynomial<109, 19> { using ZPZ = aerobus::zpz<109>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                                                                                               ZPZV<0>
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<103»; }; //</pre>
                              NOLINT
04499
                                                template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
                             POLYV<ZPZV<1>, ZPZV<110»; }; // NOLINT
04500
                                                  template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
                              POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3»; }; // NOLINT
 04501
                                             template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<110»; }; // NOLINT template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
 04502
                             POLYY<ZPZY<1>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>; ZPZV<6>, ZPZV<6 , ZPZV<6
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110»; }; // NOLINT
                                               template<> struct ConwayPolynomial<113, 6> { using ZPZ = aerobus::zpz<113>; using type =
 04504
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<50>, ZPZV<30>, ZPZV<71>, ZPZV<71>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<113, 7> { using ZPZ = aerobus::zpz<113>; using type =
04505
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<110»; }; // NOLINT
                                                template<> struct ConwayPolynomial<113, 8> { using ZPZ = aerobus::zpz<113>; using type =
04506
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<98>, ZPZV<38>, ZPZV<28>, ZPZV<3»; };
                              NOLINT
04507
                                              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<71>, ZPZV<7110»; };
                               // NOLINT
                                                template<> struct ConwayPolynomial<113, 10> { using ZPZ = aerobus::zpz<113>; using type
04508
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<56>,
                              ZPZV<3»; }; // NOLINT</pre>
04509
                                              template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04510
                                                 template<> struct ConwayPolynomial<113, 12> { using ZPZ = aerobus::zpz<113>; using type =
                             POLYY<ZPZV<1>, ZPZV<2>, ZPZV<3>, ZPZV<3>, ZPZV<5>, ZPZV<5 , ZPZV<5
                                                                                                                                                                                            // NOLINT
                                                 template<> struct ConwayPolynomial<113, 13> { using ZPZ = aerobus::zpz<113>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<44>, ZPZV<110»; }; // NOLINT
  template<> struct ConwayPolynomial<113, 17> { using ZPZ = aerobus::zpz<113>; using type =
04512
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              template<> struct ConwayPolynomial<113, 19> { using ZPZ = aerobus::zpz<113>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<2>, ZPZV<22</pre>
                              NOLINT
                                                template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
                             POLYV<ZPZV<1>, ZPZV<124»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
                             POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
 04516
                             POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZV<124»; }; // NOLINT template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type =
04517
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<97>, ZPZV<3»; }; // NOLINT
                                               template<> struct ConwayPolynomial<127, 5> { using ZPZ = aerobus::zpz<127>; using type =
 04518
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<124»; }; // NOLINT template<> struct ConwayPolynomial<127, 6> { using ZPZ = aerobus::zpz<127>; using type = POLYV<2PZV<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 = DLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; using ZPZ = aerobus::zpz<127>; using type = DLYV<2PZV<0>
04519
 04520
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<14*; }; // NOLINT
                                                template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<104>, ZPZV<55>, ZPZV<5>, ZPZV<8>, ZPZV<3»; };
                             template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<119>, ZPZV<126>, ZPZV<124»;</pre>
```

```
}; // NOLINT
template<> struct ConwayPolynomial<127, 10> { using ZPZ = aerobus::zpz<127>; using type =
04523
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<107>, ZPZV<64>, ZPZV<95>, ZPZV<60>, ZPZV<4>,
                            ZPZV<3»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<124»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<127, 12> { using ZPZ = aerobus::zpz<127>; using type
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<119>, ZPZV<25>, ZPZV<33>, ZPZV<97>, ZPZV<15>, ZPZV<99>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<8 , ZPZV<8 ,
04526
                                           template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type =
                            \texttt{POLYV} < \texttt{ZPZV} < \texttt{0} >, \ \texttt{ZPZV} < \texttt{
                            ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<124»; };</pre>
                                                                                                                                                                                                                        // 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<0>, ZPZV<0>, ZPZV<9>, ZPZV<124»; }; // NOLINT
template<> struct ConwayPolynomial<127, 19> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0</pre>
                                                                                                                                                                                                                                                                                                                                                                                                                                ZPZV<0>,
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>, ZPZV<30>, ZPZV<30</pre>
                                             template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
                          POLYV<ZPZV<1>, ZPZV<129»; }; // NOLINT
                                           template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
04530
                         POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2»; }; // NOLINT
                                             template<> struct ConwayPolynomial<131, 3> { using ZPZ = aerobus::zpz<131>; using type =
 04531
                           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<129»; }; // NOLINT template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type =
 04532
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<131, 5> { using ZPZ = aerobus::zpz<131>; using type =
 04533
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<12>, ZPZV<12>; // NOLINT template<> struct ConwayPolynomial<131, 6> { using ZPZ = aerobus::zpz<131>; using type =
04534
                          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 =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<129»; };
                           template<> struct ConwayPolynomial<131, 8> { using ZPZ = aerobus::zpz<131>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<72>, ZPZV<116>, ZPZV<104>, ZPZV<2»; }; //
                           NOLINT
                                             template<> struct ConwayPolynomial<131, 9> { using ZPZ = aerobus::zpz<131>; using type =
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<19>, ZPZV<129»; };
04538
                                           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
04539
                                             template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                     // NOLINT
                            ZPZV<6>, ZPZV<129»; };</pre>
04540
                                           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<50>, ZPZV<122>, ZPZV<40>, ZPZV<83>, ZPZV<125>,
                            ZPZV<28>, ZPZV<103>, ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type =
04541
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<129»; }; // NOLINT</pre>
04542
                                          template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                             template<> struct ConwayPolynomial<131, 19> { using ZPZ = aerobus::zpz<131>; using type :
04543
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>
04544
                                             template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
                          POLYV<ZPZV<1>, ZPZV<134»; }; // NOLINT
                                            template<> struct ConwayPolynomial<137, 2> { using ZPZ = aerobus::zpz<137>; using type =
04545
                          POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3»; }; // NOLINT
                                             template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134»; }; // NOLINT
                                           template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
 04547
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<95>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
04548
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<134»; }; // NOLINT
 04549
                                              template<> struct ConwayPolynomial<137, 6> { using ZPZ = aerobus::zpz<137>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<116>, ZPZV<102>, ZPZV<3>, ZPZV<3»; }; // NOLINT
 04550
                                          template<> struct ConwayPolynomial<137, 7> { using ZPZ = aerobus::zpz<137>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<134*; }; // NOLINT template<> struct ConwayPolynomial<137, 8> { using ZPZ = aerobus::zpz<137>; using type =
 04551
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105>, ZPZV<21>, ZPZV<34>, ZPZV<34>, ZPZV<39»; }; //
                                           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<0>, ZPZV<1>, ZPZV<80>, ZPZV<122>, ZPZV<134»;
                            }; // NOLINT
04553
                                            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<33>, ZPZV<119>,
                           ZPZV<3»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<137, 11> { using ZPZ = aerobus::zpz<137>; using type
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<1>, ZPZV<134»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<137, 12> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<61>, ZPZV<40>, ZPZV<40>, ZPZV<12>, ZPZV<36>,
 04555
```

```
ZPZV<135>, ZPZV<61>, ZPZV<3»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<137, 13> { using ZPZ = aerobus::zpz<137>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04557
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<4>, ZPZV<134»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<137, 19> { using ZPZ = aerobus::zpz<137>; using type
04558
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<134»; }; //</pre>
                                NOLINT
                                                   template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
04559
                               POLYV<ZPZV<1>, ZPZV<137»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<139, 2> { using ZPZ = aerobus::zpz<139>; using type =
                               POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2»; }; // NOLINT
 04561
                                                   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 =
04562
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<139, 5> { using ZPZ = aerobus::zpz<139>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<137»; }; // NOLINT
 04564
                                                   template<> struct ConwayPolynomial<139, 6> { using ZPZ = aerobus::zpz<139>; using type =
                                \verb|Polyv<zpzv<1>, & zpzv<0>, & zpzv<4>, & zpzv<46>, & zpzv<10>, & zpzv<118>, & zpzv<2»; & ; & // & NoLINT & | NoLINT & 
                             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<3>, ZPZV<3 , ZPZV<3 
04565
                                                    template<> struct ConwayPolynomial<139, 8> { using ZPZ = aerobus::zpz<139>; using type =
                                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>, ZPZV<36>, ZPZV<21>, ZPZV<2»; };
                               NOLINT
04567
                                                   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<87>, ZPZV<8137»; };
                                // NOLINT
04568
                                                     template<> struct ConwayPolynomial<139, 10> { using ZPZ = aerobus::zpz<139>; using type =
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<110>, ZPZV<48>, ZPZV<130>, ZPZV<66>, ZPZV<106>, ZPZV<2»; }; // NOLINT
                              \label{eq:convayPolynomial} $$ template<> struct ConwayPolynomial<139, 11> { using ZPZ = aerobus::zpz<139>; using type = POLYV<ZPZV<1>, ZPZV<0>, 
                                ZPZV<7>, ZPZV<137»; };</pre>
                                                                                                                                                         // NOLINT
                                                    template<> struct ConwayPolynomial<139, 12> { using ZPZ = aerobus::zpz<139>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<120>, ZPZV<75>, ZPZV<41>, ZPZV<41>, ZPZV<77>, ZPZV<106>,
                                ZPZV<8>, ZPZV<10>, ZPZV<2»; };</pre>
                                                                                                                                                                                                   // NOLINT
04571
                                                  template<> struct ConwayPolynomial<139, 13> { using ZPZ = aerobus::zpz<139>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                    template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type =
04572
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                                                                                                  // NOLINT
04573
                                                  template<> struct ConwayPolynomial<139, 19> { using ZPZ = aerobus::zpz<139>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04574
                                                    template<> struct ConwayPolynomial<149, 1> { using ZPZ = aerobus::zpz<149>; using type =
                               POLYV<ZPZV<1>, ZPZV<147»; }; // NOLINT
 04575
                                                   template<> struct ConwayPolynomial<149, 2> { using ZPZ = aerobus::zpz<149>; using type =
                              POLYV<ZPZV<1>, ZPZV<145>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<149, 3> { using ZPZ = aerobus::zpz<149>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147»; ); // NOLINT template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<2»; }; // NOLINT
 04578
                                                    template<> struct ConwayPolynomial<149, 5> { using ZPZ = aerobus::zpz<149>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<147»; }; // NOLINT
04579
                                                  template<> struct ConwayPolynomial<149, 6> { using ZPZ = aerobus::zpz<149>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<15>, ZPZV<15>, ZPZV<55>, ZPZV<55>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<149, 7> { using ZPZ = aerobus::zpz<149>; using type
 04580
                              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<147»; }; //
04581
                                                  template<> struct ConwayPolynomial<149, 8> { using ZPZ = aerobus::zpz<149>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<140>, ZPZV<25>, ZPZV<123>, ZPZV<123>, ZPZV<2»; };
                               NOLINT
04582
                                                   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<0>, ZPZV<146>, ZPZV<20>, ZPZV<147»;
                               }; // NOLINT
                                                   template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type =
04583
                                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 =
04584
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                 template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<91>, ZPZV<52>, ZPZV<9>,
                                ZPZV<104>, ZPZV<110>, ZPZV<2»; }; // NOLINT</pre>
04586
                                                  template<> struct ConwayPolynomial<149, 13> { using ZPZ = aerobus::zpz<149>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<147»; };</pre>
                                                                                                                                                                                                                                                         // NOLINT
                                                   template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type
                               POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<29>, ZPZV<147»; }; // NOLINT
    template<> struct ConwayPolynomial<149, 19> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
04588
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<147»; }; //</pre>
04589
                                      template<> struct ConwayPolynomial<151, 1> { using ZPZ = aerobus::zpz<151>; using type =
                       POLYV<ZPZV<1>, ZPZV<145»; }; // NOLINT
04590
                                        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 =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<145»; }; // NOLINT
                                       template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
04592
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6»; }; // NOLINT
                                        template<> struct ConwayPolynomial<151, 5> { using ZPZ = aerobus::zpz<151>; using type =
04593
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<145»; }; // NOLINT
                                        template<> struct ConwayPolynomial<151, 6> { using ZPZ = aerobus::zpz<151>; using type =
04594
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<125>, ZPZV<18>, ZPZV<15>, ZPZV<6»; }; // NOLINT
04595
                                      template<> struct ConwayPolynomial<151, 7> { using ZPZ = aerobus::zpz<151>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<445»; }; // NOLINT template<> struct ConwayPolynomial<151, 8> { using ZPZ = aerobus::zpz<151>; using type =
04596
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<140>, ZPZV<12>, ZPZV<43>, ZPZV<43>; };
                                        template<> struct ConwayPolynomial<151, 9> { using ZPZ = aerobus::zpz<151>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<126>, ZPZV<96>, ZPZV<145»;
                        }; // NOLINT
04598
                                       template<> struct ConwayPolynomial<151, 10> { using ZPZ = aerobus::zpz<151>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<21>, ZPZV<104>, ZPZV<49>, ZPZV<20>, ZPZV<142>,
                        ZPZV<6»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<151, 11> { using ZPZ = aerobus::zpz<151>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<1>, ZPZV<145»; }; // NOLINT</pre>
04600
                                         template<> struct ConwayPolynomial<151, 12> { using ZPZ = aerobus::zpz<151>; using type :
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<121>, ZPZV<101>, ZPZV<6>, ZPZV<77>, ZPZV<107>, ZPZV<147>, ZPZV<14
04601
                                         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
04602
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<145»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<151, 19> { using ZPZ = aerobus::zpz<151>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<145»; }; //</pre>
                        NOLINT
04604
                                       template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
                       POLYV<ZPZV<1>, ZPZV<152»; }; // NOLINT
04605
                                       template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
                       POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5»; }; // NOLINT
04606
                                        template<> struct ConwayPolynomial<157, 3> { using ZPZ = aerobus::zpz<157>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<152»; }; // NOLINT template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
04607
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<136>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<157, 5> { using ZPZ = aerobus::zpz<157>; using type =
04608
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<152»; }; // NOLINT
                                        template<> struct ConwayPolynomial<157, 6> { using ZPZ = aerobus::zpz<157>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<130>, ZPZV<43>, ZPZV<144>, ZPZV<5»; }; // NOLINT
                       template<> struct ConwayPolynomial<157, 7> { using ZPZ = aerobus::zpz<157>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; }; // NOLINT
04610
                                        template<> struct ConwayPolynomial<157, 8> { using ZPZ = aerobus::zpz<157>; using type
04611
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<97>, ZPZV<40>, ZPZV<153>, ZPZV<5»; };
                       template<> struct ConwayPolynomial<157, 9> { using ZPZ = aerobus::zpz<157>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<114>, ZPZV<52>, ZPZV<152»;
                        }; // NOLINT
                                        template<> struct ConwayPolynomial<157, 10> { using ZPZ = aerobus::zpz<157>; using type =
04613
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<61>, ZPZV<22>, ZPZV<124>, ZPZV<61>, ZPZV<93>,
                        ZPZV<5»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<157, 11> { using ZPZ = aerobus::zpz<157>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<29>, ZPZV<152»; }; // NOLINT</pre>
04615
                                       template<> struct ConwayPolynomial<157, 12> { using ZPZ = aerobus::zpz<157>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110>, ZPZV<72>, ZPZV<137>, ZPZV<43>,
                        ZPZV<152>, ZPZV<57>, ZPZV<5»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<157, 13> { using ZPZ = aerobus::zpz<157>; using type :
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<156>, ZPZV<9>, ZPZV<152»; }; // NOLINT
template<> struct ConwayPolynomial<157, 17> { using ZPZ = aerobus::zpz<157>; using type
04617
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<152»; };</pre>
                                      template<> struct ConwayPolynomial<157, 19> { using ZPZ = aerobus::zpz<157>; using type
04618
                        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<152»; }; //</pre>
                        NOLINT
04619
                                        template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
                       POLYV<ZPZV<1>, ZPZV<161»; };
                                                                                                                                              // NOLINT
                                         template<> struct ConwayPolynomial<163, 2> { using ZPZ = aerobus::zpz<163>; using type =
                       POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; }; // NOLINT
04621
                                      template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<161»; }; // NOLINT template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
04622
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2»; };
                                              template<> struct ConwayPolynomial<163, 5> { using ZPZ = aerobus::zpz<163>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<161»; }; // NOLINT
                                            template<> struct ConwayPolynomial<163, 6> { using ZPZ = aerobus::zpz<163>; using type =
04624
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<83>, ZPZV<25>, ZPZV<156>, ZPZV<2»; }; // NOLINT
04625
                                            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
 04626
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<132>, ZPZV<83>, ZPZV<6>, ZPZV<6>, ZPZV<2»; }; //
04627
                                           template<> struct ConwayPolynomial<163, 9> { using ZPZ = aerobus::zpz<163>; using type =
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<15>, ZPZV<162>, ZPZV<121>, ZPZV<161»;
                           }; // NOLINT
template<> struct ConwayPolynomial<163, 10> { using ZPZ = aerobus::zpz<163>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<111>, ZPZV<120>, ZPZV<125>, ZPZV<15>, ZPZV<0>,
                            ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<163, 11> { using ZPZ = aerobus::zpz<163>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<31>, ZPZV<38>, ZPZV<103>, ZPZV<10>, ZPZV<30>, ZPZV
                                            template<> struct ConwayPolynomial<163, 13> { using ZPZ = aerobus::zpz<163>; using type =
04631
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<161»; }; // NOLINT
                                              template<> struct ConwayPolynomial<163, 17> { using ZPZ = aerobus::zpz<163>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<161»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<163, 19> { using ZPZ = aerobus::zpz<163>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>
                           NOLINT
                                              template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
                           POLYV<ZPZV<1>, ZPZV<162»; }; // NOLINT
 04635
                                             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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162»; }; // NOLINT
04636
 04637
                                              template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<167, 5> { using ZPZ = aerobus::zpz<167>; using type =
 04638
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<162»; }; // NOLINT template<> struct ConwayPolynomial<167, 6> { using ZPZ = aerobus::zpz<167>; using type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<38>, ZPZV<2>, ZPZV<5»; }; // NOLINT
 04639
                                              template<> struct ConwayPolynomial<167, 7> { using ZPZ = aerobus::zpz<167>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<162»; };
04641
                                           template<> struct ConwayPolynomial<167, 8> { using ZPZ = aerobus::zpz<167>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<149>, ZPZV<56>, ZPZV<113>, ZPZV<5»; }; //
                           NOLINT
                                            template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type =
04642
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<165>, ZPZV<162»;
                           }; // NOLINT
04643
                                             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
04644
                                             template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<24>, ZPZV<162»; };</pre>
                                                                                                                                     // NOLINT
                           template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<142>, ZPZV<10>, ZPZV<142>, ZPZV<131>,
                           ZPZV<140>, ZPZV<41>, ZPZV<57>, ZPZV<5>»; }; // NOLINT
template<> struct ConwayPolynomial<167, 13> { using ZPZ = aerobus::zpz<167>; using type :
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<167, 17> { using ZPZ = aerobus::zpz<167>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<162»; }; // NOLINT
template<> struct ConwayPolynomial<167, 19> { using ZPZ = aerobus::zpz<167>; using type =
04648
                           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<14>, ZPZV<162»; }; //</pre>
                                           template<> struct ConwayPolynomial<173, 1> { using ZPZ = aerobus::zpz<173>; using type =
                           POLYV<ZPZV<1>, ZPZV<171»; }; // NOLINT
                                             template<> struct ConwayPolynomial<173, 2> { using ZPZ = aerobus::zpz<173>; using type =
04650
                           POLYV<ZPZV<1>, ZPZV<169>, ZPZV<2»; }; // NOLINT
                                              template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
 04652
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<2»; }; // NOLINT
                                             template<> struct ConwayPolynomial<173, 5> { using ZPZ = aerobus::zpz<173>; using type =
04653
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<171»; }; // NOLINT
 04654
                                             template<> struct ConwayPolynomial<173, 6> { using ZPZ = aerobus::zpz<173>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<134>, ZPZV<107>, ZPZV<2»; }; // NOLINI
 04655
                                           template<> struct ConwayPolynomial<173, 7> { using ZPZ = aerobus::zpz<173>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 8> { using ZPZ = aerobus::zpz<173>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<125>, ZPZV<158>, ZPZV<27>, ZPZV<2»; }; //
 04656
```

```
NOLINT
04657
                                             template<> struct ConwayPolynomial<173, 9> { using ZPZ = aerobus::zpz<173>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<104>, ZPZV<171»;
                            }; // NOLINT
04658
                                                template<> struct ConwayPolynomial<173, 10> { using ZPZ = aerobus::zpz<173>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<16>, ZPZV<16>, ZPZV<16>, ZPZV<106>, ZPZV<58>, ZPZV<2»; }; // NOLINT
                                               template<> struct ConwayPolynomial<173, 11> { using ZPZ = aerobus::zpz<173>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<12>, ZPZV<171»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type =
04660
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<64>, ZPZV<46>, ZPZV<46>, ZPZV<166>, ZPZV<0>,
                            ZPZV<159>, ZPZV<22>, ZPZV<2»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<173, 13> { using ZPZ = aerobus::zpz<173>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171»; }; // NOLINT
template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 7>, \ \texttt{ZPZV} < 171 »; \ \}; \ // \ \texttt{NOLINT} 
                                             template<> struct ConwayPolynomial<173, 19> { using ZPZ = aerobus::zpz<173>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6</pre>, //
                            NOLINT
                                             template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
04664
                            POLYV<ZPZV<1>, ZPZV<177»; }; // NOLINT
                                              template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
                            POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2»; }; // NOLINT
04666
                                             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 =
04667
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<109>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<179, 5> { using ZPZ = aerobus::zpz<179>; using type =
04668
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<177»; }; // NOLINT
04669
                                            template<> struct ConwayPolynomial<179, 6> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<91>, ZPZV<55>, ZPZV<10>>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
04670
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<177»; }; // NOLINT template<> struct ConwayPolynomial<179, 8> { using ZPZ = aerobus::zpz<179>; using type =
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<144>, ZPZV<73>, ZPZV<2»; };
04672
                                            template<> struct ConwayPolynomial<179, 9> { using ZPZ = aerobus::zpz<179>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<50>, ZPZV<50>
                            // NOLINT
04673
                                               template<> struct ConwayPolynomial<179, 10> { using ZPZ = aerobus::zpz<179>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<115>, ZPZV<71>, ZPZV<150>, ZPZV<49>, ZPZV<87>,
                            ZPZV<2»; }; // NOLINT</pre>
04674
                                             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>
                                               template<> struct ConwayPolynomial<179, 12> { using ZPZ = aerobus::zpz<179>; using type
04675
                            POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<103>, ZPZV<83>, ZPZV<43>, ZPZV<46>, ZPZV<76>, ZPZV<8>, ZPZV<177>, ZPZV<1>, ZPZV<1
                                            template<> struct ConwayPolynomial<179, 13> { using ZPZ = aerobus::zpz<179>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              template<> struct ConwayPolynomial<179,
04677
                                                                                                                                                                                                                                           17> { using ZPZ = aerobus::zpz<179>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<177»; }; // NOLINT
template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                            NOLINT
04679
                                               template<> struct ConwayPolynomial<181, 1> { using ZPZ = aerobus::zpz<181>; using type =
                            POLYV<ZPZV<1>, ZPZV<179»; }; // NOLINT
                                            template<> struct ConwayPolynomial<181, 2> { using ZPZ = aerobus::zpz<181>; using type =
                           POLYV<ZPZV<1>, ZPZV<177>, ZPZV<2»; }; // NOLINT
04681
                                               template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179»; }; // NOLINT template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
04682
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<105>, ZPZV<2»; }; // NOLINT
                                             template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
04683
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<179»; }; // NOLINT
04684
                                              template<> struct ConwayPolynomial<181, 6> { using ZPZ = aerobus::zpz<181>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<163>, ZPZV<169>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type
04685
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<179»; }; //
                                            template<> struct ConwayPolynomial<181, 8> { using ZPZ = aerobus::zpz<181>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<108>, ZPZV<22>, ZPZV<149>, ZPZV<2»; }; //
                            NOLINT
04687
                           template<> struct ConwayPolynomial<181, 9> { using ZPZ = aerobus::zpz<181>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<107>, ZPZV<168>, ZPZV<179»;</pre>
                          }; // NOLINT
template<> struct ConwayPolynomial<181, 10> { using ZPZ = aerobus::zpz<181>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<88>,
                                             template<> struct ConwayPolynomial<181, 11> { using ZPZ = aerobus::zpz<181>; using type =
04689
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
```

```
ZPZV<24>, ZPZV<179»; };</pre>
                                  template<> struct ConwayPolynomial<181, 12> { using ZPZ = aerobus::zpz<181>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<141>, ZPZV<45>, ZPZV<122>,
                    ZPZV<175>, ZPZV<12>, ZPZV<10>, ZPZV<2»; }; // NOLINT
   template<> struct ConwayPolynomial<181, 13> { using ZPZ = aerobus::zpz<181>; using type :
04691
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  template<> struct ConwayPolynomial<181, 17> { 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<9>, ZPZV<179»; }; // NOLINT
    template<> struct ConwayPolynomial<181, 19> { using ZPZ = aerobus::zpz<181>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04693
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<36</pre>
04694
                                template<> struct ConwayPolynomial<191, 1> { using ZPZ = aerobus::zpz<191>; using type =
                   POLYV<ZPZV<1>, ZPZV<172»; }; // NOLINT template<> struct ConwayPolynomial<191, 2> { using ZPZ = aerobus::zpz<191>; using type =
04695
                   POLYV<ZPZV<1>, ZPZV<190>, ZPZV<190»; }; // NOLINT template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172»; }; // NOLINT
                                  template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; }; // NOLINT
                                template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
04698
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<172»; }; // NOLINT
04699
                                  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
 04700
                                template<> struct ConwayPolynomial<191, 7> { using ZPZ = aerobus::zpz<191>; using type =
                   POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<172»; }; // NOLINT template<> struct ConwayPolynomial<191, 8> { using ZPZ = aerobus::zpz<191>; using type =
 04701
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<164>, ZPZV<139>, ZPZV<171>, ZPZV<19»; }; //
                    NOLINT
04702
                                  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<6>, ZPZV<62>, ZPZV<124>, ZPZV<172»;
                    }; // NOLINT
                    template<> struct ConwayPolynomial<191, 10> { using ZPZ = aerobus::zpz<191>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<113>, ZPZV<47>, ZPZV<173>, ZPZV<74>,
04703
                    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
                    ZPZV<6>, ZPZV<172»; };</pre>
                                                                                                // NOLINT
04705
                                template<> struct ConwayPolynomial<191, 12> { using ZPZ = aerobus::zpz<191>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<168>, ZPZV<25>, ZPZV<49>, ZPZV<90, ZPZV<7>, ZPZV<151>, ZPZV<19»; }; // NOLINT
                                 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>
04707
                                template<> struct ConwayPolynomial<191, 17> { using ZPZ = aerobus::zpz<191>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<172»; }; // NOLINT template<> struct ConwayPolynomial<191, 19> { using ZPZ = aerobus::zpz<191>; using type =
04708
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<190>, ZPZV<190>, ZPZV<172»; }; //</pre>
04709
                                 template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
                   POLYV<ZPZV<1>, ZPZV<188»; }; // NOLINT
                                  template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
04710
                    POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5»; }; // NOLINT
                               template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
 04711
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; }; // NOLINT
template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
 04712
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<648, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
04713
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<188»; }; // NOLINT
                                  template<> struct ConwayPolynomial<193, 6> { using ZPZ = aerobus::zpz<193>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<8>, ZPZV<172>, ZPZV<5»; }; // NOLINT
                                template<> struct ConwayPolynomial<193, 7> { using ZPZ = aerobus::zpz<193>; using type =
04715
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<88, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 8> { using ZPZ = aerobus::zpz<193>; using type =
04716
                    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 =
04717
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<168>, ZPZV<27>, ZPZV<188»;
                    }; // NOLINT
04718
                                  template<> struct ConwayPolynomial<193, 10> { using ZPZ = aerobus::zpz<193>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<51>, ZPZV<77>, ZPZV<78>, ZPZV<89>,
                     ZPZV<5»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; }; // NOLINT
04720
                               template<> struct ConwayPolynomial<193, 12> { using ZPZ = aerobus::zpz<193>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<52>, ZPZV<135>, ZPZV<135>, ZPZV<152>,
                    ZPZV<90>, ZPZV<46>, ZPZV<28>, ZPZV<5»; }; // NOLINT</pre>
                                template<> struct ConwayPolynomial<193, 13> { using ZPZ = aerobus::zpz<193>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                    ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<188»; }; // NOLINT
   template<> struct ConwayPolynomial<193, 17> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<188»; };</pre>
                         template<> struct ConwayPolynomial<193, 19> { using ZPZ = aerobus::zpz<193>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          \texttt{ZPZV} < \texttt{0>, ZPZV} < \texttt{0
                         NOLINT
04724
                                         template<> struct ConwavPolynomial<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 =
 04725
                         POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2»; }; // NOLINT
04726
                                       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 =
04727
                       POLYV<ZPZV<1>, ZPZV<16>, ZPZV<124, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<197, 5> { using ZPZ = aerobus::zpz<197>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<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 =
 04729
04730
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<195»; }; // NOLINT
                                         template<> struct ConwayPolynomial<197, 8> { using ZPZ = aerobus::zpz<197>; using type =
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<176>, ZPZV<96>, ZPZV<29>, ZPZV<2»; };
                         NOLINT
04732
                                       template<> struct ConwayPolynomial<197, 9> { using ZPZ = aerobus::zpz<197>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<127>, ZPZV<8>, ZPZV<195»;
                         }; // NOLINT
                                           template<> struct ConwayPolynomial<197, 10> { using ZPZ = aerobus::zpz<197>; using type
                         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<14>, ZPZV<195»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<197, 12> { using ZPZ = aerobus::zpz<197>; using type =
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<168>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>,
                         ZPZV<90>, ZPZV<163>, ZPZV<2»; }; // NOLINT</pre>
                        \label{eq:convayPolynomial} $$ \text{template} > \text{struct ConwayPolynomial} < 197, 13> { using ZPZ = aerobus::zpz<197>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
04736
                         ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<195»; }; // NOLINT
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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<195»; }; // NOLINT</pre>
04738
                                       template<> struct ConwayPolynomial<197, 19> { using ZPZ = aerobus::zpz<197>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
                       POLYV<ZPZV<1>, ZPZV<196»; }; // NOLINT
 04740
                                       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 =
04741
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; }; // NOLINT template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; }; // NOLINT
 04743
                                        template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<31>6>; // NOLINT template<> struct ConwayPolynomial<199, 6> { using ZPZ = aerobus::zpz<199>; using type =
04744
                        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
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; };
                                       template<> struct ConwayPolynomial<199, 8> { using ZPZ = aerobus::zpz<199>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<160>, ZPZV<23>, ZPZV<159>, ZPZV<3»; }; //
                         NOLINT
                                       template<> struct ConwayPolynomial<199, 9> { using ZPZ = aerobus::zpz<199>; using type =
04747
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<177>, ZPZV<141>, ZPZV<196»;
                         }; // NOLINT
04748
                                       template<> struct ConwayPolynomial<199, 10> { using ZPZ = aerobus::zpz<199>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<171>, ZPZV<158>, ZPZV<31>, ZPZV<54>, ZPZV<9>,
                         ZPZV<3»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<199, 11> { using ZPZ = aerobus::zpz<199>; using type =
04749
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<1>, ZPZV<196»; };</pre>
                                                                                                                          // NOLINT
                                         template<> struct ConwayPolynomial<199, 12> { using ZPZ = aerobus::zpz<199>; using type =
                         POLYV<ZPZV<1>, 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 :
04751
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<196»; };</pre>
                                                                                                                                                                                                    // NOLINT
                                       template<> struct ConwayPolynomial<199,
                                                                                                                                                                                                            17> { using ZPZ = aerobus::zpz<199>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<196»; }; // NOLINT
template<> struct ConwayPolynomial<199, 19> { using ZPZ = aerobus::zpz<199>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04753
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19*, 
 04754
                                        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 =
 04755
                         POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; }; // NOLINT
```

```
04756
                                    template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<209»; }; // NOLINT template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
04757
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2»; }; // NOLINT
                                  template<> struct ConwayPolynomial<211, 5> { using ZPZ = aerobus::zpz<211>; using type =
 04758
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<209; }; // NOLINT template<> struct ConwayPolynomial<211, 6> { using ZPZ = aerobus::zpz<211>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<81>, ZPZV<194>, ZPZV<133>, ZPZV<2»; }; // NOLINI
 04760
                                 template<> struct ConwayPolynomial<211, 7> { using ZPZ = aerobus::zpz<211>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<209»; }; // NOLINT
                                 template<> struct ConwayPolynomial<211, 8> { using ZPZ = aerobus::zpz<211>; using type =
04761
                     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 =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<139>, ZPZV<26>, ZPZV<209»;
                     }; // NOLINT
                                   template<> struct ConwayPolynomial<211, 10> { using ZPZ = aerobus::zpz<211>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<61>, ZPZV<148, ZPZV<87>, ZPZV<125>,
                     ZPZV<2»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<211, 11> { using ZPZ = aerobus::zpz<211>; using type :
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<7>, ZPZV<209%; }; // NOLINT
  template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type =
04765
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<50>, ZPZV<145>, ZPZV<126>, ZPZV<184>,
                     ZPZV<84>, ZPZV<27>, ZPZV<2»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<211, 13> { using ZPZ = aerobus::zpz<211>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<209»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<211,
                                                                                                                                                                             17> { using ZPZ = aerobus::zpz<211>; using type
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<209; }; // NOLINT template<> struct ConwayPolynomial<211, 19> { using ZPZ = aerobus::zpz<211>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<209»; }; //</pre>
                     NOLINT
                                   template<> struct ConwayPolynomial<223, 1> { using ZPZ = aerobus::zpz<223>; using type =
04769
                    POLYV<ZPZV<1>, ZPZV<220»; }; // NOLINT
                                   template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
                     POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; }; // NOLINT
                                  template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
04772
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<223, 5> { using ZPZ = aerobus::zpz<223>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; }; // NOLINT
 04774
                                  template<> struct ConwayPolynomial<223, 6> { using ZPZ = aerobus::zpz<223>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68>, ZPZV<24>, ZPZV<196>, ZPZV<3»; }; // NOLINT
                                 template<> struct ConwayPolynomial<223, 7> { using ZPZ = aerobus::zpz<223>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<220»; }; // NOLINT
                                 template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
04776
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<3»; }; //
04777
                                 template<> struct ConwayPolynomial<223, 9> { using ZPZ = aerobus::zpz<223>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<164>, ZPZV<64>, ZPZV<220»;
                     }; // NOLINT
04778
                     template<> struct ConwayPolynomial<223, 10> { using ZPZ = aerobus::zpz<223>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<118>, ZPZV<177>, ZPZV<87>, ZPZV<99>, ZPZV<62>,
                     ZPZV<3»: }: // NOLINT
                                  template<> struct ConwayPolynomial<223, 11> { using ZPZ = aerobus::zpz<223>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                     ZPZV<8>, ZPZV<220»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<223, 12> { using ZPZ = aerobus::zpz<223>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<64>, ZPZV<64>, ZPZV<94>, ZPZV<11>, ZPZV<64>,
                     ZPZV<151>, ZPZV<213>, ZPZV<3»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<223, 13> { using ZPZ = aerobus::zpz<223>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<220»; }; // NOLINT
template<> struct ConwayPolynomial<223, 17> { using ZPZ = aerobus::zpz<223>; using type =
04782
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; };</pre>
                                                                                                                                                                                                                                                                                                    // NOLINT
                                  template<> struct ConwayPolynomial<223, 19> { using ZPZ = aerobus::zpz<223>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>
                     NOLINT
                                  template<> struct ConwayPolynomial<227, 1> { using ZPZ = aerobus::zpz<227>; using type =
                     POLYV<ZPZV<1>, ZPZV<225»; }; // NOLINT
 04785
                                  template<> struct ConwayPolynomial<227, 2> { using ZPZ = aerobus::zpz<227>; using type =
                     POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
 04786
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<225»; }; // NOLINT
template<> struct ConwayPolynomial<227, 4> { using ZPZ = aerobus::zpz<227>; using type =
 04787
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2»; };
                                                                                                                                                                                                                         // NOLINT
                                   template<> struct ConwayPolynomial<227, 5> { using ZPZ = aerobus::zpz<227>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<225»; }; // NOLINT
 04789
                                 template<> struct ConwayPolynomial<227, 6> { using ZPZ = aerobus::zpz<227>; using type =
                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<24>, ZPZV<135>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<227, 7> { using ZPZ = aerobus::zpz<227>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<225»; };
                               template<> struct ConwayPolynomial<227, 8> { using ZPZ = aerobus::zpz<227>; using type
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<151>, ZPZV<176>, ZPZV<106>, ZPZV<2»; }; //
                  NOLINT
                              template<> struct ConwayPolynomial<227, 9> { using ZPZ = aerobus::zpz<227>; using type
04792
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<183>, ZPZV<225»;
                  }; // NOLINT
                             template<> struct ConwayPolynomial<227, 10> { using ZPZ = aerobus::zpz<227>; using type
04793
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<129>, ZPZV<12>, ZPZV<93>, ZPZV<77>,
                  ZPZV<2»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type =
04794
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  ZPZV<2>, ZPZV<225»; };</pre>
                                                                                            // NOLINT
                               template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<123>, ZPZV<99>, ZPZV<160>, ZPZV<96>,
                  ZPZV<127>, ZPZV<142>, ZPZV<94>, ZPZV<2»; }; // NOLINT
  template<> struct ConwayPolynomial<227, 13> { using ZPZ = aerobus::zpz<227>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                 // NOLINT
                  template<> struct ConwayPolynomial<227, 19> { using ZPZ = aerobus::zpz<227>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04798
                  NOLINT
                              template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
04799
                  POLYV<ZPZV<1>, ZPZV<223»; }; // NOLINT
04800
                               template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
                  POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»; }; // NOLINT
                              template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
04801
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; }; // NOLINT template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
04802
                  04803
                               template<> struct ConwayPolynomial<229, 5> { using ZPZ = aerobus::zpz<229>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223»; }; // NOLINT
                              template<> struct ConwayPolynomial<229, 6> { using ZPZ = aerobus::zpz<229>; using type =
04804
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<160>, ZPZV<186>, ZPZV<6»; }; // NOLINT
04805
                               template<> struct ConwayPolynomial<229, 7> { using ZPZ = aerobus::zpz<229>; using type
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<223»; }; // NOLINT
04806
                             template<> struct ConwayPolynomial<229, 8> { using ZPZ = aerobus::zpz<229>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<62>, ZPZV<62>, ZPZV<60>, ZPZV<6»; }; //
                  NOLINT
04807
                              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
04808
                              template<> struct ConwayPolynomial<229, 10> { using ZPZ = aerobus::zpz<229>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<185>, ZPZV<135>, ZPZV<158>, ZPZV<167>, ZPZV<98>, ZPZV<6»; }; // NOLINT
                               template<> struct ConwayPolynomial<229, 11> { using ZPZ = aerobus::zpz<229>; using type
04809
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  ZPZV<2>, ZPZV<223»; }; // NOLINT</pre>
04810
                             template<> struct ConwayPolynomial<229, 12> { using ZPZ = aerobus::zpz<229>; using type :
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<131>, ZPZV<140>, ZPZV<25>, ZPZV<6>, ZPZV<172>, ZPZV<9>, ZPZV<6>; }; // NOLINT
04811
                               template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type :
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                  ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<223»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type =
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<223»; }; // NOLINT
    template<> struct ConwayPolynomial<229, 19> { using ZPZ = aerobus::zpz<229>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04813
                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<228>, ZPZV<15>, ZPZV<223»; }; //</pre>
04814
                             template<> struct ConwayPolynomial<233, 1> { using ZPZ = aerobus::zpz<233>; using type =
                  POLYV<ZPZV<1>, ZPZV<230»; }; // NOLINT
                              template<> struct ConwayPolynomial<233, 2> { using ZPZ = aerobus::zpz<233>; using type =
04815
                 POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
                               template<> struct ConwayPolynomial<233, 3> { using ZPZ = aerobus::zpz<233>; using type =
04816
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
04817
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<45>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<233, 5> { using ZPZ = aerobus::zpz<233>; using type =
04818
                 POLYYCZPZV<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
04820
                             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<0>, ZPZV<4>, ZPZV<4>, ZPZV<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 8> { using ZPZ = aerobus::zpz<233>; using type =
04821
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<202>, ZPZV<135>, ZPZV<181>, ZPZV<3»; }; //
                             template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<146>, ZPZV<230»;
                  }; // NOLINT
                               template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type =
04823
                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<28>, ZPZV<71>, ZPZV<102>, ZPZV<3>, ZPZV<48>,
```

```
ZPZV<3»; };</pre>
                                          template<> struct ConwayPolynomial<233, 11> { using ZPZ = aerobus::zpz<233>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<5>, ZPZV<230»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<233, 12> { using ZPZ = aerobus::zpz<233>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<114>, ZPZV<31>, ZPZV<19>,
ZPZV<216>, ZPZV<20>, ZPZV<3»; }; // NOLINT</pre>
04825
                                            template<> struct ConwayPolynomial<233, 13> { using ZPZ = aerobus::zpz<233>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<230»; }; // NOLINT
  template<> struct ConwayPolynomial<233, 17> { using ZPZ = aerobus::zpz<233>; using type =
04827
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43, ZPZV<230»; }; // NOLINT
template<> struct ConwayPolynomial<233, 19> { using ZPZ = aerobus::zpz<233>; using type
                          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<25>, ZPZV<25>, ZPZV<230»; }; //</pre>
                          NOLINT
                                          template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
04829
                          POLYV<ZPZV<1>, ZPZV<232»; }; // NOLINT
                                          template<> struct ConwayPolynomial<239, 2> { using ZPZ = aerobus::zpz<239>; using type =
                          POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7»; }; // NOLINT
 04831
                                           template<> struct ConwayPolynomial<239, 3> { using ZPZ = aerobus::zpz<239>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<232»; }; // NOLINT template<> struct ConwayPolynomial<239, 4> { using ZPZ = aerobus::zpz<239>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7»; }; // NOLINT
 04832
                                           template<> struct ConwayPolynomial<239, 5> { using ZPZ = aerobus::zpz<239>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; }; // NOLINT
 04834
                                          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 =
04835
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<232»; }; // NOLINT
                                           template<> struct ConwayPolynomial<239, 8> { using ZPZ = aerobus::zpz<239>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<202>, ZPZV<54>, ZPZV<7»; };
                         template<> struct ConwayPolynomial<239, 9> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<28>, ZPZV<288>, ZPZV<232»; };</pre>
04837
                           // NOLINT
                                           template<> struct ConwayPolynomial<239, 10> { using ZPZ = aerobus::zpz<239>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<68>, ZPZV<226>, ZPZV<127>,
                           ZPZV<108>, ZPZV<7»; };</pre>
                                                                                                                            // NOLINT
04839
                                         template<> struct ConwayPolynomial<239, 11> { using ZPZ = aerobus::zpz<239>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<8>, ZPZV<232»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<239, 12> { using ZPZ = aerobus::zpz<239>; using type =
04840
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<14>, ZPZV<113>, ZPZV<182>, ZPZV<101>, ZPZV<81>, ZPZV<216>, ZPZV<7»; }; // NOLINT
04841
                                         template<> struct ConwayPolynomial<239, 13> { using ZPZ = aerobus::zpz<239>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           template<> struct ConwayPolynomial<239, 17> { using ZPZ = aerobus::zpz<239>; using type =
04842
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<232»; };</pre>
04843
                                         template<> struct ConwayPolynomial<239, 19> { using ZPZ = aerobus::zpz<239>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<232»; }; //</pre>
                          NOLINT
                                            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 =
                          POLYV<ZPZV<1>, ZPZV<238>, ZPZV<7»; }; // NOLINT
                                         template<> struct ConwayPolynomial<241, 3> { using ZPZ = aerobus::zpz<241>; using type =
04846
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<234»; }; // NOLINT template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
 04847
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<15>, ZPZV<18>, // NOLINT template<> struct ConwayPolynomial<241, 5> { using ZPZ = aerobus::zpz<241>; using type =
04848
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<234»; }; // NOLINT
                        template<> struct ConwayPolynomial<241, 6> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<63>, ZPZV<6>, ZPZV<5>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<241, 7> { using ZPZ = aerobus::zpz<241>; using type =
 04849
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<234»; }; //
 04851
                                          template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<113>, ZPZV<212>, ZPZV<153>, ZPZV<7»; }; //
                          NOLINT
04852
                                          template<> struct ConwayPolynomial<241, 9> { using ZPZ = aerobus::zpz<241>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<236>, ZPZV<125>, ZPZV<234»;
                                         template<> struct ConwayPolynomial<241, 10> { using ZPZ = aerobus::zpz<241>; using type :
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<27>, ZPZV<145>, ZPZV<168>, ZPZV<55>,
                          ZPZV<7»: }: // NOLINT
04854
                                         template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<3>, ZPZV<234»; };</pre>
                                                                                                                                 // NOLINT
                                         template<> struct ConwayPolynomial<241, 12> { using ZPZ = aerobus::zpz<241>; using type :
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<10>, ZPZV<109>, ZPZV<168>, ZPZV<22>,
                          <code>ZPZV<197></code>, <code>ZPZV<17></code>, <code>ZPZV<7»</code>; <code>}</code>; <code>// NOLINT</code>
                          template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
 04856
```

```
ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234»; };</pre>
                                                                                                                                                                                                                        // NOLINT
                                              template<> struct ConwayPolynomial<241, 17> { using ZPZ = aerobus::zpz<241>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<234»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<241, 19> { using ZPZ = aerobus::zpz<241>; using type
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
04858
                                                                                                                                                                                                                                                                                                                                                                                                                                7.P.7.V<0>.
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234»; }; //</pre>
                                              template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
04859
                           POLYV<ZPZV<1>, ZPZV<245»; }; // NOLINT
                                              template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
04860
                           POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; }; // NOLINT
                                              template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
04861
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT
 04862
                                           template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<200>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<251, 5> { using ZPZ = aerobus::zpz<251>; using type =
04863
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<25; }; // NOLINT template<> struct ConwayPolynomial<251, 6> { using ZPZ = aerobus::zpz<251>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<247>, ZPZV<151>, ZPZV<179>, ZPZV<6»; }; // NOLINT
                                              template<> struct ConwayPolynomial<251, 7> { using ZPZ = aerobus::zpz<251>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<245»; }; // NOLINT
04866
                                            template<> struct ConwayPolynomial<251, 8> { using ZPZ = aerobus::zpz<251>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<142>, ZPZV<215>, ZPZV<173>, ZPZV<6*; }; //
                           NOLINT
                                              template<> struct ConwayPolynomial<251, 9> { using ZPZ = aerobus::zpz<251>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<106>, ZPZV<245»;
                            }; // NOLINT
04868
                                              template<> struct ConwayPolynomial<251, 10> { using ZPZ = aerobus::zpz<251>; using type =
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3, ZPZV<138>, ZPZV<110>, ZPZV<45>, ZPZV<34>, ZPZV<34>, ZPZV<149>, ZPZV<6»; }; // NOLINT
04869
                                              template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<26>, ZPZV<245»; };</pre>
                                                                                                                                      // NOLINT
                           template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<192>, ZPZV<53>, ZPZV<20>, ZPZV<20>, ZPZV<15>,
04870
                            ZPZV<201>, ZPZV<232>, ZPZV<6»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<251, 13> { using ZPZ = aerobus::zpz<251>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<245»; }; // NOLINT</pre>
04872
                                           template<> struct ConwayPolynomial<251,
                                                                                                                                                                                                                                17> { using ZPZ = aerobus::zpz<251>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04873
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>
04874
                                            template<> struct ConwayPolynomial<257, 1> { using ZPZ = aerobus::zpz<257>; using type =
                           POLYV<ZPZV<1>, ZPZV<254»; }; // NOLINT
                                              template<> struct ConwayPolynomial<257, 2> { using ZPZ = aerobus::zpz<257>; using type =
 04875
                           POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3»; }; // NOLINT
                                              template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<254»; }; // NOLINT template<> struct ConwayPolynomial<257, 4> { using ZPZ = aerobus::zpz<257>; using type =
 04877
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<18>, ZPZV<3*; }; // NOLINT template<> struct ConwayPolynomial<257, 5> { using ZPZ = aerobus::zpz<257>; using type =
 04878
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<254»; }; // NOLINT
                                           template<> struct ConwayPolynomial<257, 6> { using ZPZ = aerobus::zpz<257>; using type =
 04879
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<18>, ZPZV<188>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type =
 04880
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<254»; }; // NOLT template<> struct ConwayPolynomial<257, 8> { using ZPZ = aerobus::zpz<257>; using type =
                                                                                                                                                                                                                                                                                                                                                                                                                         // NOLINT
04881
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<179>, ZPZV<140>, ZPZV<162>, ZPZV<3»; }; //
                                           template<> struct ConwayPolynomial<257, 9> { using ZPZ = aerobus::zpz<257>; using type =
04882
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<201>, ZPZV<50>, ZPZV<254»;
                            }; // NOLINT
04883
                                              template<> struct ConwayPolynomial<257, 10> { using ZPZ = aerobus::zpz<257>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<225>, ZPZV<26>, ZPZV<20>,
                            ZPZV<3»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<257, 11> { using ZPZ = aerobus::zpz<257>; using type :
04884
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<40>, ZPZV<254»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<257, 12> { using ZPZ = aerobus::zpz<257>; using type =
04885
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<23>, ZPZV<225>, ZPZV<215>, ZPZV<2173>, ZPZV<249>, ZPZV<148>, ZPZV<20>, ZPZV<3»; }; // NOLINT
                                           template<> struct ConwayPolynomial<257, 13> { using ZPZ = aerobus::zpz<257>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           template<> struct ConwayPolynomial<257, 17> { using ZPZ = aerobus::zpz<257>; using type =
04887
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<254»; };</pre>
                                            template<> struct ConwayPolynomial<257, 19> { using ZPZ = aerobus::zpz<257>; using type
                           POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<254»; }; //</pre>
                            NOLINT
 04889
                                             template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<258»; };
                      template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
             POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
04891
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; }; // NOLINT template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
04892
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; }; // NOLINT
04893
                      template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<258»; }; // NOLINT
04894
                     template<> struct ConwayPolynomial<263, 6> { using ZPZ = aerobus::zpz<263>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<250>, ZPZV<25>, ZPZV<25>; Using type = DIVV<ZPZV<1>, ZPZV<0>, ZPZV<263, ZPZV<250>, ZPZV<250>, ZPZV<263>; Using type = DIVV<ZPZV<1 - ZPZV<263, ZPZV<263, ZPZV<263>; using type = DIVV<ZPZV<1 - ZPZV<1 
04895
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<258»; };
                      template<> struct ConwayPolynomial<263, 8> { using ZPZ = aerobus::zpz<263>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<227>, ZPZV<170>, ZPZV<7>, ZPZV<5»; };
                     template<> struct ConwayPolynomial<263, 9> { using ZPZ = aerobus::zpz<263>; using type =
04897
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<261>, ZPZV<29>, ZPZV<258»;
                       template<> struct ConwayPolynomial<263, 10> { using ZPZ = aerobus::zpz<263>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<245>, ZPZV<231>, ZPZV<198>, ZPZV<145>,
             ZPZV<119>, ZPZV<5»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<263, 11> { using ZPZ = aerobus::zpz<263>; using type =
04899
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
             ZPZV<2>, ZPZV<258»; };</pre>
                                                                 // NOLINT
                      template<> struct ConwayPolynomial<263, 12> { using ZPZ = aerobus::zpz<263>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<174>, ZPZV<162>, ZPZV<252>,
              ZPZV<47>, ZPZV<45>, ZPZV<180>, ZPZV<5»; }; // NOLINT</pre>
04901
                      template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
             POLYV<ZPZV<1>, ZPZV<267»; }; // NOLINT
                     template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
04902
             POLYV<ZPZV<1>, ZPZV<268>, ZPZV<2»; }; // NOLINT
                      template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
04903
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<267»; }; // NOLINT
                      template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
04904
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2»; }; // NOLINT
                     template<> struct ConwayPolynomial<269, 5> { using ZPZ = aerobus::zpz<269>; using type =
04905
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<267»; }; // NOLINT
04906
                      template<> struct ConwayPolynomial<269, 6> { using ZPZ = aerobus::zpz<269>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<206>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<269, 7> { using ZPZ = aerobus::zpz<269>; using type =
04907
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>; // NOLINT template<> struct ConwayPolynomial<269, 8> { using ZPZ = aerobus::zpz<269>; using type =
04908
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<220>, ZPZV<131>, ZPZV<232>, ZPZV<23x; }; //
04909
                      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<21+>, ZPZV<267>, ZPZV<267>,
             }; // NOLINT
04910
                     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 =
04911
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
             ZPZV<20>, ZPZV<267»; }; // NOLINT
             template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<165>, ZPZV<165>, ZPZV<215>, ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2»; }; // NOLINT
04912
                     template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type =
04913
             POLYV<ZPZV<1>, ZPZV<265»; }; // NOLINT
04914
                      template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
            POLYV<ZPZV<1>, ZPZV<269>, ZPZV<69; }; // NOLINT
template<> struct ConnayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
04915
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6»; }; // NOLINT
                     template<> struct ConwayPolynomial<271, 5> { using ZPZ = aerobus::zpz<271>; using type =
04917
             template<> struct ConwayPolynomial<271, 6> { using ZPZ = aerobus::zpz<271>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<207>, ZPZV<207>, ZPZV<81>, ZPZV<6»; }; // NOLINT
04918
                                                                                                            7> { using ZPZ = aerobus::zpz<271>; using type
04919
                      template<> struct ConwayPolynomial<271,
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT template<> struct ConwayPolynomial<271, 8> { using ZPZ = aerobus::zpz<271>; using type =
04920
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<114>, ZPZV<69>, ZPZV<68; };
             NOLINT
                     template<> struct ConwayPolynomial<271, 9> { using ZPZ = aerobus::zpz<271>; using type =
04921
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<266>, ZPZV<186>, ZPZV<265»;
             }; // NOLINT
04922
                      template<> struct ConwayPolynomial<271, 10> { using ZPZ = aerobus::zpz<271>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<1>, ZPZV<133>, ZPZV<10>, ZPZV<256>, ZPZV<74>,
             ZPZV<126>, ZPZV<6»; }; // NOLINT</pre>
                      template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type :
04923
              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<265»; }; // NOLINT</pre>
04924
                      template<> struct ConwayPolynomial<271, 12> { using ZPZ = aerobus::zpz<271>; using type =
             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162>, ZPZV<210>, ZPZV<116>, ZPZV<205>, ZPZV<237>, ZPZV<256>, ZPZV<130>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
04925
```

```
POLYV<ZPZV<1>, ZPZV<272»; };
                        template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
              POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; }; // NOLINT
                       template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
04927
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; }; // NOLINT template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
04928
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<5»; }; // NOLINT
04929
                        template<> struct ConwayPolynomial<277, 5> { using ZPZ = aerobus::zpz<277>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<272»; }; // NOLINT
                       template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type =
04930
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<9>, ZPZV<118>, ZPZV<5»; }; // NOLINT
                       template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type =
04931
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<159>, ZPZV<176>, ZPZV<5»; }; //
                       template<> struct ConwayPolynomial<277, 9> { using ZPZ = aerobus::zpz<277>; using type =
04933
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4777>, ZPZV<110>, ZPZV<272»;
              }; // NOLINT
                         template<> struct ConwayPolynomial<277, 10> { using ZPZ = aerobus::zpz<277>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<206>, ZPZV<253>, ZPZV<237>, ZPZV<241>,
              ZPZV<260>, ZPZV<5»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type =
04935
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
              ZPZV<5>, ZPZV<272»; };</pre>
                                                                       // NOLINT
                         template<> struct ConwayPolynomial<277, 12> { using ZPZ = aerobus::zpz<277>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<183>, ZPZV<218>, ZPZV<240>, ZPZV<40>,
               ZPZV<180>, ZPZV<115>, ZPZV<202>, ZPZV<5»; }; // NOLINT</pre>
04937
                         template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
              POLYV<ZPZV<1>, ZPZV<278»; }; // NOLINT
                       template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
04938
              POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; }; // NOLINT
                        template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<278»; }; // NOLINT
                        template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
04940
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3»; }; // NOLINT
                        template<> struct ConwayPolynomial<281, 5> { using ZPZ = aerobus::zpz<281>; using type =
04941
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<278»; }; // NOLINT
04942
                         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 template<> struct ConwayPolynomial<281, 7> { using ZPZ = aerobus::zpz<281>; using type =
04943
             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<28*; }; // NOLINT template<> struct ConwayPolynomial<281, 8> { using ZPZ = aerobus::zpz<281>; using type =
04944
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195>, ZPZV<279>, ZPZV<140>, ZPZV<3»; }; //
              NOLINT
04945
                        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<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
              }; // NOLINT
04946
                        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»; };</pre>
                                                                        // NOLINT
                        template<> struct ConwayPolynomial<281, 11> { using ZPZ = aerobus::zpz<281>; using type :
04947
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
              ZPZV<36>, ZPZV<278»; }; // NOLINT
                        template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type =
04948
              POLYVCZPZVC1>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC2>, ZPZVC68>, ZPZVC103>, ZPZVC116>, ZPZVC58>, ZPZVC28>, ZPZVC191>, ZPZVC3»; }; // NOLINT
                      template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
04949
              POLYV<ZPZV<1>, ZPZV<280»; }; // NOLINT
04950
                        template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
             POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<283, 3> { using ZPZ = aerobus::zpz<283>; using type =
04951
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
                         template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; }; // NOLINT
04953
                       template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
             template<> struct ConwayPolynomial<283, 6> { using ZPZ = aerobus::zpz<283>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<68>, ZPZV<73>, ZPZV<3»; }; // NOLINT
04954
04955
                        template<> struct ConwayPolynomial<283,
                                                                                                                        7> { using ZPZ = aerobus::zpz<283>; using type
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<80»; }; // NOLIN template<> struct ConwayPolynomial<283, 8> { using ZPZ = aerobus::zpz<283>; using type =
04956
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<179>, ZPZV<32>, ZPZV<232>, ZPZV<33»; }; //
              NOLINT
                       template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type
04957
              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<136>, ZPZV<280»;
              }; // NOLINT
04958
                        template<> struct ConwayPolynomial<283, 10> { using ZPZ = aerobus::zpz<283>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<271>, ZPZV<185>, ZPZV<68>, ZPZV<100>,
              ZPZV<219>, ZPZV<3»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<283, 11> { using ZPZ = aerobus::zpz<283>; using type :
04959
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
              ZPZV<4>, ZPZV<280»; }; // NOLINT
04960
                        template<> struct ConwayPolynomial<283, 12> { using ZPZ = aerobus::zpz<283>; using type :
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<8>, ZPZV<96>, ZPZV<229>, ZPZV<49>, ZPZV<14>, ZPZV<56>, ZPZV<3»; }; // NOLINT
04961
                      template<> struct ConwayPolynomial<293, 1> { using ZPZ = aerobus::zpz<293>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<291»; };
                           template<> struct ConwayPolynomial<293, 2> { using ZPZ = aerobus::zpz<293>; using type =
                POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; }; // NOLINT
                           template<> struct ConwayPolynomial<293, 3> { using ZPZ = aerobus::zpz<293>; using type =
04963
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT template<> struct ConwayPolynomial<293, 4> { using ZPZ = aerobus::zpz<293>; using type =
04964
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2»; }; // NOLINT
                           template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
04965
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
04966
                          template<> struct ConwayPolynomial<293, 6> { using ZPZ = aerobus::zpz<293>; using type =
               04967
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<291»; };
                           template<> struct ConwayPolynomial<293, 8> { using ZPZ = aerobus::zpz<293>; using type
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<
                          template<> struct ConwayPolynomial<293, 9> { using ZPZ = aerobus::zpz<293>; using type =
04969
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20, ZPZV<208>, ZPZV<290>, ZPZV<291»;
                            template<> struct ConwayPolynomial<293, 10> { using ZPZ = aerobus::zpz<293>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<186>, ZPZV<28>, ZPZV<46>, ZPZV<184>, ZPZV<24>,
                ZPZV<2»; }; // NOLINT</pre>
                          \texttt{template} <> \texttt{struct ConwayPolynomial} < 293, 11> \{ \texttt{using ZPZ = aerobus::zpz} < 293>; \texttt{using type = aerobus::zpz} 
04971
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                ZPZV<3>, ZPZV<291»; };</pre>
                                                                                 // NOLINT
                            template<> struct ConwayPolynomial<293, 12> { using ZPZ = aerobus::zpz<293>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<210>, ZPZV<210>, ZPZV<125>, ZPZV<212>,
                 ZPZV<167>, ZPZV<144>, ZPZV<157>, ZPZV<2»; }; // NOLINT</pre>
04973
                           template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
                POLYV<ZPZV<1>, ZPZV<302»; }; // NOLINT
                          template<> struct ConwayPolynomial<307, 2> { using ZPZ = aerobus::zpz<307>; using type =
04974
                POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5»; }; // NOLINT
                           template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<302»; }; // NOLINT
                           template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
04976
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5»; }; // NOLINT
04977
                           template<> struct ConwayPolynomial<307, 5> { using ZPZ = aerobus::zpz<307>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<302»; }; // NOLINT
04978
                           template<> struct ConwayPolynomial<307, 6> { using ZPZ = aerobus::zpz<307>; using type =
                POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<213>, ZPZV<172>, ZPZV<61>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<307, 7> { using ZPZ = aerobus::zpz<307>; using type =
04979
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04980
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<283>, ZPZV<232>, ZPZV<131>, ZPZV<5»; }; //
04981
                           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<1>, ZPZV<165>, ZPZV<70>, ZPZV<302»;
                }; // NOLINT
04982
                           template<> struct ConwavPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
                POLYV<ZPZV<1>, ZPZV<294»; }; // NOLINT
04983
                           template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
                POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; }; // NOLINT
04984
                          template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; }; // NOLINT template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
04985
               POLYV<ZPZV<1>, ZPZV<3>, ZPZV<163>, ZPZV<163>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294»; }; // NOLINT
                           template<> struct ConwayPolynomial<311, 6> { using ZPZ = aerobus::zpz<311>; using type =
04987
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<167>, ZPZV<152>, ZPZV<17»; }; // NOLINT
04988
                          template<> struct ConwayPolynomial<311, 7> { using ZPZ = aerobus::zpz<311>; using type =
                POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294*; }; // NOLINT
                          template<> struct ConwayPolynomial<311, 8> { using ZPZ = aerobus::zpz<311>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<162>, ZPZV<118>, ZPZV<2>, ZPZV<217»; }; //
                template<> struct ConwayPolynomial<311, 9> { using ZPZ = aerobus::zpz<311>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<287>, ZPZV<287>, ZPZV<294»;</pre>
04990
                }; // NOLINT
                           template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
04991
                POLYV<ZPZV<1>, ZPZV<303»; }; // NOLINT
                           template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
04992
                POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; }; // NOLINT
                           template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
04993
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; }; // NOLINT
                template<> struct ConwayPolynomial
template struct ConwayPolynomial
template struct ConwayPolynomial
for a polynomial
template struct ConwayPolynomial
template struct ConwayPolynomial
template
template
struct ConwayPolynomial
template
struct ConwayPolynomial
template
template
struct ConwayPolynomial
template
struct ConwayPolynomial
template
struct ConwayPolynomial
struct ConwayPolynomial
template
struct ConwayPolynomial
template
struct ConwayPolynomial
struct Conwa
04994
                          template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
04995
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<303»; }; // NOLINT
04996
                           template<> struct ConwayPolynomial<313, 6> { using ZPZ = aerobus::zpz<313>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<196>, ZPZV<213>, ZPZV<253>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<313, 7> { using ZPZ = aerobus::zpz<313>; using type
04997
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<303»; };
                          template<> struct ConwayPolynomial<313, 8> { using ZPZ = aerobus::zpz<313>; using type
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<306>, ZPZV<99>, ZPZV<106>, ZPZV<10»; }; //
                template<> struct ConwayPolynomial<313, 9> { using ZPZ = aerobus::zpz<313>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<267>, ZPZV<300>, ZPZV<303»;</pre>
04999
```

```
}; // NOLINT
0.5000
           template<> struct ConwayPolynomial<317, 1> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<315»; }; // NOLINT
           template<> struct ConwayPolynomial<317, 2> { using ZPZ = aerobus::zpz<317>; using type =
05001
      POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
05002
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<315»; // NOLINT
           template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
05003
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<317, 5> { using ZPZ = aerobus::zpz<317>; using type =
05004
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<156>, { using ZPZ = aerobus: ZPZ<31/7, using type = POLYV<ZPZV<1, ZPZV<0>, ZPZV<0, ZPZV<0>, ZPZV<0>, ZPZV<2), ZPZV<2>, ZPZV<315»; }; // NOLINT = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<195>, ZPZV<156>, ZPZV<4>, ZPZV<2»; }; // NOLINT
05005
           template<> struct ConwayPolynomial<317, 7> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<315»; }; // NOLINT
05007
          template<> struct ConwayPolynomial<317, 8> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<207>, ZPZV<85>, ZPZV<31>, ZPZV<2»; };
      NOLINT
           template<> struct ConwayPolynomial<317, 9> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<284>, ZPZV<296>, ZPZV<315»;
      }; // NOLINT
05009
           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 =
05010
      POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
05011
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<328»; }; // NOLINT
           template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
05012
      template<> struct ConwayPolynomial<331, 5> { using ZPZ = aerobus::zpz<331>; using type =
05013
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<328»; }; // NOLINT
05014
           template<> struct ConwayPolynomial<331, 6> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<205>, ZPZV<159>, ZPZV<3»; }; // NOLINT
05015
          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<5>, ZPZV<328»; }; // NOLINT template<> struct ConwayPolynomial<331, 8> { using ZPZ = aerobus::zpz<331>; using type =
05016
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2), ZPZV<249>, ZPZV<308>, ZPZV<78>, ZPZV<3»; }; //
      NOLINT
05017
           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
05018
           template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<327»; }; // NOLINT
05019
           template<> struct ConwayPolynomial<337, 2> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<332>, ZPZV<10»; }; // NOLINT
05020
           template<> struct ConwayPolynomial<337, 3> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
05021
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<337, 5> { using ZPZ = aerobus::zpz<337>; using type =
05022
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT
           template<> struct ConwayPolynomial<337, 6> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<216>, ZPZV<127>, ZPZV<109>, ZPZV<10»; }; // NOLINT
05024
          template<> struct ConwayPolynomial<337, 7> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<327»; }; // NOLINT
           template<> struct ConwayPolynomial<337, 8> { using ZPZ = aerobus::zpz<337>; using type
05025
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<331>, ZPZV<246>, ZPZV<251>, ZPZV<251>, ZPZV<10»; }; //
      template<> struct ConwayPolynomial<337, 9> { using ZPZ = aerobus::zpz<337>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<148>, ZPZV<98>, ZPZV<327»;
      }; // NOLINT
05027
           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 =
      POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
05029
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
05030
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
05031
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<345»; }; // NOLINT
05032
          template<> struct ConwayPolynomial<347, 6> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<34>, ZPZV<26>, ZPZV<56>, ZPZV<28; }; // NOLINT template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
05033
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<45»; };
                                                                                                     // NOLINT
           template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<187>, ZPZV<213>, ZPZV<117>, ZPZV<2»; };
05035
          template<> struct ConwayPolynomial<347, 9> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<255>, ZPZV<345»;
       }; // NOLINT
           template<> struct ConwayPolynomial<349, 1> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<347»; }; // NOLINT
05037
           template<> struct ConwayPolynomial<349, 2> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; }; // NOLINT
05038
```

```
05039
            template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<349, 5> { using ZPZ = aerobus::zpz<349>; using type =
05040
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<347»; }; // NOLINT
       template<> struct ConwayPolynomial<349, 6> { using ZPZ = aerobus::zpz<349>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<135>, ZPZV<177>, ZPZV<316>, ZPZV<2»; }; // NOLINT</pre>
05041
            template<> struct ConwayPolynomial<349, 7> { using ZPZ = aerobus::zpz<349>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<14>, ZPZV<347»; }; //
           template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type =
05043
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<328>, ZPZV<268>, ZPZV<2»; }; //
       NOLINT
           template<> struct ConwayPolynomial<349, 9> { using ZPZ = aerobus::zpz<349>; using type =
05044
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<290>, ZPZV<130>, ZPZV<347»;
       }; // NOLINT
05045
            template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type =
       POLYY<ZPZV<1>, ZPZV<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
05046
       POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3»; }; // NOLINT
            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
05049
           template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<350»; }; // NOLINT
05050
            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
05051
           template<> struct ConwayPolynomial<353, 7> { using ZPZ = aerobus::zpz<353>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 8> { using ZPZ = aerobus::zpz<353>; using type =
05052
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<182>, ZPZV<26>, ZPZV<37>, ZPZV<3»; };
       NOLINT
05053
            template<> struct ConwayPolynomial<353, 9> { using ZPZ = aerobus::zpz<353>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<319>, ZPZV<49>, ZPZV<350»;
       }; // NOLINT
05054
            template<> struct ConwayPolynomial<359, 1> { using ZPZ = aerobus::zpz<359>; using type =
       POLYV<ZPZV<1>, ZPZV<352»; }; // NOLINT
            template<> struct ConwayPolynomial<359, 2> { using ZPZ = aerobus::zpz<359>; using type =
05055
       POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7»; }; // NOLINT
05056
            template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<352, ZPZY<352, }; // NOLINT template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
05057
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<22>, ZPZV<229>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<359, 5> { using ZPZ = aerobus::zpz<359>; using type =
05058
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; }; // NOLINT
            template<> struct ConwayPolynomial<359, 6> { using ZPZ = aerobus::zpz<359>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<309>, ZPZV<327>, ZPZV<327>, ZPZV<7»; }; // NOLINT
05060
           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<1>, ZPZV<352»; }; // NOLINT
05061
           template<> struct ConwayPolynomial<359, 8> { using ZPZ = aerobus::zpz<359>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<301>, ZPZV<143>, ZPZV<271>, ZPZV<7»; }; //
            template<> struct ConwayPolynomial<359, 9> { using ZPZ = aerobus::zpz<359>; using type =
05062
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<356>, ZPZV<165>, ZPZV<352»;
       }; // NOLINT
            template<> struct ConwayPolynomial<367, 1> { using ZPZ = aerobus::zpz<367>; using type =
05063
       POLYV<ZPZV<1>, ZPZV<361»; }; // NOLINT
            template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
       POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6»; }; // NOLINT
            template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
05065
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<361»; }; // NOLINT

template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6»; }; // NOLINT

template<> struct ConwayPolynomial<367, 5> { using ZPZ = aerobus::zpz<367>; using type =
05066
05067
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<361»; }; // NOLINT
05068
           template<> struct ConwayPolynomial<367, 6> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<222>, ZPZV<321>, ZPZV<324>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type
05069
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<361»; }; // NOLINT
           template<> struct ConwayPolynomial<367, 8> { using ZPZ = aerobus::zpz<367>; using type =
05070
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<335>, ZPZV<282>, ZPZV<50>, ZPZV<66; };
05071
           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
05072
            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 =
05073
       POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2»; }; // NOLINT
05074
           template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<371»; }; // NOLINT
template<> struct ConwayPolynomial<373, 4> { using ZPZ = aerobus::zpz<373>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304>, ZPZV<2»; }; // NOLINT
05075
            template<> struct ConwayPolynomial<373, 5> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<371»; }; // NOLINT
05077
           template<> struct ConwayPolynomial<373, 6> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<83>, ZPZV<108>, ZPZV<2»; }; // NOLINT 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<7>, ZPZV<371»; };
                  template<> struct ConwayPolynomial<373, 8> { using ZPZ = aerobus::zpz<373>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<203>, ZPZV<219>, ZPZV<66>, ZPZV<2»; }; //
          NOLINT
05080
                 template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<238>, ZPZV<370>, ZPZV<371»;
05081
                  template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
          POLYV<ZPZV<1>, ZPZV<377»; }; // NOLINT
                 template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
05082
          POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<379, 3> { using ZPZ = aerobus::zpz<379>; using type =
05083
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<5>, ZPZY<5>, ZPZY<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
05085
                 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
                 template<> struct ConwayPolynomial<379, 6> { using ZPZ = aerobus::zpz<379>; using type =
05086
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<374>, ZPZV<364>, ZPZV<246>, ZPZV<2°, }; // NOLINT
                 template<> struct ConwayPolynomial<379,
                                                                                         7> { using ZPZ = aerobus::zpz<379>; using type
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<377»; }; //
05088
                 template<> struct ConwayPolynomial<379, 8> { using ZPZ = aerobus::zpz<379>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<210>, ZPZV<194>, ZPZV<173>, ZPZV<2»; }; //
          NOLINT
05089
                 template<> struct ConwayPolynomial<379, 9> { using ZPZ = aerobus::zpz<379>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<362>, ZPZV<369>, ZPZV<377»;
          }; // NOLINT
05090
                 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 =
05091
          POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; }; // NOLINT
05092
                  template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<378»; }; // NOLINT template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
05093
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<383, 5> { using ZPZ = aerobus::zpz<383>; using type =
05094
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
                  template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<8>, ZPZV<158>, ZPZV<5»; };
                 template<> struct ConwayPolynomial<383, 7> { using ZPZ = aerobus::zpz<383>; using type =
05096
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 - , ZPZ
05097
                 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»; }; //
                 template<> struct ConwayPolynomial<383, 9> { using ZPZ = aerobus::zpz<383>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137>, ZPZV<76>, ZPZV<378»;
           }; // NOLINT
05099
                  template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
          POLYV<ZPZV<1>, ZPZV<387»; }; // NOLINT
05100
                 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 =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387»; }; // NOLINT template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
05102
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<266>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
05103
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<387»; }; // NOLINT
                 template<> struct ConwayPolynomial<389, 6> { using ZPZ = aerobus::zpz<389>; using type =
05104
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<339>, ZPZV<255>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<389, 7> { using ZPZ = aerobus::zpz<389>; using type :
05105
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<28, ZPZV<287; // NOLINT template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::zpz<389>; using type =
05106
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<351>, ZPZV<19>, ZPZV<290>, ZPZV<2»; };
05107
                 template<> struct ConwayPolynomial<389, 9> { using ZPZ = aerobus::zpz<389>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<308>, ZPZV<387»;
           }; // NOLINT
                 \texttt{template<> struct ConwayPolynomial<397, 1> \{ using ZPZ = aerobus:: zpz<397>; using type = 200 t
05108
          POLYV<ZPZV<1>, ZPZV<392»; }; // NOLINT
05109
                  template<> struct ConwayPolynomial<397, 2> { using ZPZ = aerobus::zpz<397>; using type =
          POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5»; }; // NOLINT
05110
                 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 =
05111
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<363; }; // NOLINT template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
05112
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392»; }; // NOLINT
05113
                  template<> struct ConwayPolynomial<397, 6> { using ZPZ = aerobus::zpz<397>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<274>, ZPZV<287>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 7> { using ZPZ = aerobus::zpz<397>; using type :
05114
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<392»; }; // NOLINT
                 template<> struct ConwayPolynomial<397, 8> { using ZPZ = aerobus::zpz<397>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<375>, ZPZV<255>, ZPZV<203>, ZPZV<5»; };
          NOLINT
05116
                 template<> struct ConwayPolynomial<397, 9> { using ZPZ = aerobus::zpz<397>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<166>, ZPZV<252>, ZPZV<392»;
          }; // NOLINT
```

```
05117
           template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<398»; }; // NOLINT
05118
          template<> struct ConwayPolynomial<401, 2> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3»; }; // NOLINT
0.5119
           template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<38*; }; // 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 =
05121
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<398»; }; // NOLINT
           template<> struct ConwayPolynomial<401, 6> { using ZPZ = aerobus::zpz<401>; using type =
05122
      POLYY<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
05123
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<398»; }; //
05124
          template<> struct ConwayPolynomial<401, 8> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<380>, ZPZV<113>, ZPZV<164>, ZPZV<3»; }; //
      NOLTNT
      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»;
05125
           template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
          template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
05127
      POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT
           template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
05128
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388»; }; // NOLINT
05129
          template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
      template<> struct ConwayPolynomial<409, 5> { using ZPZ = aerobus::zpz<409>; using type =
05130
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<388»; }; // NOLINT
          template<> struct ConwayPolynomial<409, 6> { using ZPZ = aerobus::zpz<409>; using type =
05131
      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 =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<388»; }; // NOLINT
05133
          template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<256>, ZPZV<69>, ZPZV<396>, ZPZV<396>, ZPZV<21»; }; //
      NOLINT
          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<211>, ZPZV<388»;
      }; // NOLINT
05135
           template<> struct ConwayPolynomial<419, 1> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<417»; }; // NOLINT
           template<> struct ConwayPolynomial<419, 2> { using ZPZ = aerobus::zpz<419>; using type =
05136
      POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type =
05138
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<373, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<419, 5> { using ZPZ = aerobus::zpz<419>; using type =
05139
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; // NOLINT
05140
           template<> struct ConwayPolynomial<419, 6> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<411>, ZPZV<33>, ZPZV<257>, ZPZV<2»; }; // NOLINT
05141
          template<> struct ConwayPolynomial<419, 7> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 8> { using ZPZ = aerobus::zpz<419>; using type =
05142
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<38>, ZPZV<388>, ZPZV<151>, ZPZV<2s; }; //
05143
          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<9>, ZPZV<386>, ZPZV<417»;
      }; // NOLINT
05144
          template<> struct ConwayPolynomial<421, 1> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<419»; }; // NOLINT
           template<> struct ConwayPolynomial<421, 2> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT
05146
          template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419»; }; // NOLINT template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
05147
      POLYVCZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2; }; // NOLINT template<> struct ConwayPolynomial<421, 5> { using ZPZ = aerobus::zpz<421>; using type =
05148
      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
05150
           template<> struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<419»; }; // NOLINT template<> struct ConwayPolynomial<421, 8> { using ZPZ = aerobus::zpz<421>; using type =
05151
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<389>, ZPZV<32>, ZPZV<77>, ZPZV<2»; };
      NOLINT
05152
          template<> struct ConwayPolynomial<421, 9> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<394>, ZPZV<345>, ZPZV<419»;
      }; // NOLINT
05153
           template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<431, 2> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7»; }; // NOLINT
05155
          template<> struct ConwayPolynomial<431, 3> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424»; }; // NOLINT template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7»; };
            template<> struct ConwayPolynomial<431, 5> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<431, 6> { using ZPZ = aerobus::zpz<431>; using type =
05158
      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
05159
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; //
           template<> struct ConwayPolynomial<431, 8> { using ZPZ = aerobus::zpz<431>; using type
05160
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<243>, ZPZV<286>, ZPZV<115>, ZPZV<7»; }; //
05161
           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*;
      }; // NOLINT
  template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
05162
      POLYV<ZPZV<1>, ZPZV<428»; }; // NOLINT
05163
           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 =
05164
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428»; }; // NOLINT
           template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<402>, ZPZV<5»; }; // NOLINT
05166
           template<> struct ConwayPolynomial<433, 5> { using ZPZ = aerobus::zpz<433>; using type =
      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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<244>, ZPZV<360>, ZPZV<360>, ZPZV<5»; }; // NOLINT
0.5167
           template<> struct ConwayPolynomial<433, 7> { using ZPZ = aerobus::zpz<433>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<428»; };
           template<> struct ConwayPolynomial<433, 8> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347>, ZPZV<32>, ZPZV<39>, ZPZV<5»; }; //
      NOLINT
           template<> struct ConwayPolynomial<433, 9> { using ZPZ = aerobus::zpz<433>; using type =
05170
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<23>, ZPZV<232>, ZPZV<45>, ZPZV<428»;
       }; // NOLINT
           template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
05171
      POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
05172
      POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
05174
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
05175
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT
05176
           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
05177
           template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT template<> struct ConwayPolynomial<439, 8> { using ZPZ = aerobus::zpz<439>; using type =
05178
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<359>, ZPZV<296>, ZPZV<266>, ZPZV<15»; }; //
       NOLINT
05179
           template<> struct ConwayPolynomial<439, 9> { using ZPZ = aerobus::zpz<439>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<342>, ZPZV<254>, ZPZV<424»;
       }; // NOLINT
05180
           template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<441»; }; // NOLINT
           template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
05181
       POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
05182
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT template<> struct ConwayPolynomial<4443, 4> { using ZPZ = aerobus::zpz<4443>; using type =
05183
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<443>; using type =
05184
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44), ZPZV<441»; }; // NOLINT
            template<> struct ConwayPolynomial<443, 6> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<298>, ZPZV<218>, ZPZV<41>, ZPZV<2»; }; // NOLINT
05186
           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<6>, ZPZV<441»; }; // NOLINT template<> struct ConwayPolynomial<443, 8> { using ZPZ = aerobus::zpz<443>; using type =
05187
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<217>, ZPZV<290>, ZPZV<29»; }; //
       NOLINT
           template<> struct ConwayPolynomial<443, 9> { using ZPZ = aerobus::zpz<443>; using type =
05188
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<125>, ZPZV<109>, ZPZV<441»;
       }; // NOLINT
05189
           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 =
      POLYV<ZPZV<1>, ZPZV<444>, ZPZV<3»; }; // NOLINT
05191
           template<> struct ConwayPolynomial<449, 3> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446»; }; // NOLINT template<> struct ConwayPolynomial<449, 4> { using ZPZ = aerobus::zpz<449>; using type =
05192
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<249>, ZPZV<249>, ZPZV<349>, ZPZV<349>, ZPZV<349>, ZPZV<349>, ZPZV<349>, ZPZV<349>, ZPZV<349>; using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<446»; }; // NOLINT
05194
           template<> struct ConwayPolynomial<449, 6> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<293>, ZPZV<69>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<449, 7> { using ZPZ = aerobus::zpz<449>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<446»; }; // NOLINT
05195
```

```
template<> struct ConwayPolynomial<449, 8> { using ZPZ = aerobus::zpz<449>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<348>, ZPZV<124>, ZPZV<33*; }; //
       NOLINT
      template<> struct ConwayPolynomial<449, 9> { using ZPZ = aerobus::zpz<449>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<226>, ZPZV<26>, ZPZV<9>, ZPZV<446»; };</pre>
05197
       // NOLINT
05198
           template<> struct ConwayPolynomial<457, 1> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<444»; }; // NOLINT
           template<> struct ConwayPolynomial<457, 2> { using ZPZ = aerobus::zpz<457>; using type =
05199
      POLYV<ZPZV<1>, ZPZV<454>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
05200
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444»; }; // NOLINT
           template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
05201
                                                                        // NOLINT
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13»; };
05202
           template<> struct ConwayPolynomial<457, 5> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44+4*; }; // NOLINT template<> struct ConwayPolynomial<457, 6> { using ZPZ = aerobus::zpz<457>; using type =
05203
      POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<205>, ZPZV<205>, ZPZV<266>, ZPZV<265>, ZPZV<267>; using ZPZ = aerobus::zpz<457>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<444»; };
           template<> struct ConwayPolynomial<457, 8> { using ZPZ = aerobus::zpz<457>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<365>, ZPZV<296>, ZPZV<412>, ZPZV<413»; }; //
      template<> struct ConwayPolynomial<457, 9> { using ZPZ = aerobus::zpz<457>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<354>, ZPZV<84>, ZPZV<444»;</pre>
05206
       }; // NOLINT
           template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
05207
      POLYV<ZPZV<1>, ZPZV<459»; }; // NOLINT
05208
           template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<461, 3> { using ZPZ = aerobus::zpz<461>; using type =
05209
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459»; }; // NOLINT
           template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
05210
      05211
           template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<459»; }; // NOLINT
      template<> struct ConwayPolynomial<461, 6> { using ZPZ = aerobus::zpz<461>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<439>, ZPZV<432>, ZPZV<329>, ZPZV<2»; }; // NOLINT
05212
05213
           template<> struct ConwayPolynomial<461,
                                                         7> { using ZPZ = aerobus::zpz<461>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<459»; };
05214
           template<> struct ConwayPolynomial<461, 8> { using ZPZ = aerobus::zpz<461>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<449>, ZPZV<321>, ZPZV<2»; }; //
       NOLINT
05215
           template<> struct ConwayPolynomial<461, 9> { using ZPZ = aerobus::zpz<461>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<210>, ZPZV<276>, ZPZV<459»;
       }; // NOLINT
05216
           template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<460»; }; // NOLINT
05217
           template<> struct ConwayPolynomial<463, 2> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; }; // NOLINT
05218
           template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<460»; }; // NOLINT
05219
           template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
05220
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<460»; }; // NOLINT
           template<> struct ConwayPolynomial<463, 6> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<462>, ZPZV<51>, ZPZV<110>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<463, 7> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<460»; }; // NOLI template<> struct ConwayPolynomial<463, 8> { using ZPZ = aerobus::zpz<463>; using type =
05223
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<234>, ZPZV<414>, ZPZV<396>, ZPZV<3»; }; //
       NOLINT
           template<> struct ConwayPolynomial<463, 9> { using ZPZ = aerobus::zpz<463>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<433>, ZPZV<433>, ZPZV<227>, ZPZV<460»;
       }; // NOLINT
05225
           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 =
05226
      POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
05227
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
05228
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<467, 5> { using ZPZ = aerobus::zpz<467>; using type =
05229
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
           template<> struct ConwayPolynomial<467, 6> { using ZPZ = aerobus::zpz<467>; using type =
05230
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<123>, ZPZV<62>, ZPZV<237>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type =
05231
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 8> { using ZPZ = aerobus::zpz<467>; using type =
05232
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<318>, ZPZV<413>, ZPZV<289>, ZPZV<2»; }; //
05233
           template<> struct ConwayPolynomial<467, 9> { using ZPZ = aerobus::zpz<467>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<397>, ZPZV<447>, ZPZV<465»;
       }; // NOLINT
05234
          template<> struct ConwayPolynomial<479, 1> { using ZPZ = aerobus::zpz<479>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<466»; };
           template<> struct ConwayPolynomial<479, 2> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; }; // NOLINT
           template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
05236
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466»; }; // NOLINT template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
05237
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13»; }; // NOLINT
           template<> struct ConwayPolynomial<479, 5> { using ZPZ = aerobus::zpz<479>; using type =
05238
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<466»; }; // NOLINT
05239
          template<> struct ConwayPolynomial<479, 6> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<243>, ZPZV<287>, ZPZV<334>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<479, 7> { using ZPZ = aerobus::zpz<479>; using type
05240
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<46»; };
           template<> struct ConwayPolynomial<479, 8> { using ZPZ = aerobus::zpz<479>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<247>, ZPZV<440>, ZPZV<17>, ZPZV<13»; }; //
          template<> struct ConwayPolynomial<479, 9> { using ZPZ = aerobus::zpz<479>; using type =
05242
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<185>, ZPZV<466»; };
          template<> struct ConwayPolynomial<487, 1> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<484»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 2> { using ZPZ = aerobus::zpz<487>; using type =
05244
      POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
05245
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3»; }; // NOLINT
05247
           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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<427>, ZPZV<185>, ZPZV<3»; }; // NOLINT
05248
05249
           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»; }; //
05250
          template<> struct ConwayPolynomial<487, 8> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<249>, ZPZV<137>, ZPZV<3»; }; //
      NOLINT
      template<> struct ConwayPolynomial<487, 9> { using ZPZ = aerobus::zpz<487>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<271>, ZPZV<4447>, ZPZV<484»;
05251
05252
           template<> struct ConwayPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<489»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
05253
      POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»: }: // NOLINT
05254
           template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type =
05255
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360>, ZPZV<2»; }; // NOLINT
          template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
05256
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 6> { using ZPZ = aerobus::zpz<491>; using type =
05257
      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 =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<489»; }; // NOLINT
05259
          template<> struct ConwayPolynomial<491, 8> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378>, ZPZV<372>, ZPZV<216>, ZPZV<2»; }; //
      NOLINT
           template<> struct ConwayPolynomial<491, 9> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<453>, ZPZV<489»;
      }; // NOLINT
05261
           template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<492»; }; // NOLINT
          template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
05262
      POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; }; // NOLINT
          template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
05264
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
05265
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<492»; }; // NOLINT
05266
           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»; };
05267
          template<> struct ConwayPolynomial<499, 7> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 8> { using ZPZ = aerobus::zpz<499>; using type =
05268
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<309>, ZPZV<200>, ZPZV<20»; }; //
          template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type
05269
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<491>, ZPZV<222>, ZPZV<492»;
      }; // NOLINT
05270
           template<> struct ConwayPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<498»; }; // NOLINT
           template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; }; // NOLINT
05272
          template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<498»; }; // NOLINT template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<32>, ZPZV<5»; }; // NOLINT
05273
```

```
template<> struct ConwayPolynomial<503, 5> { using ZPZ = aerobus::zpz<503>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<498»; }; // NOLINT
05275
           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<5s; }; // NOLINT template<> struct ConwayPolynomial<503, 7> { using ZPZ = aerobus::zpz<503>; using type
05276
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<498»; };
                                                                                                            // NOLINT
           template<> struct ConwayPolynomial<503, 8> { using ZPZ = aerobus::zpz<503>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<441>, ZPZV<203>, ZPZV<316>, ZPZV<35»; }; //
       template<> struct ConwayPolynomial<503, 9> { using ZPZ = aerobus::zpz<503>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<158>, ZPZV<337>, ZPZV<498»;</pre>
05278
       }; // NOLINT
05279
            template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<507»; }; // NOLINT
05280
           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 =
05281
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
            template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
           template<> struct ConwayPolynomial<509, 6> { using ZPZ = aerobus::zpz<509>; using type =
05284
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<350>, ZPZV<232>, ZPZV<41>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<509, 7> { using ZPZ = aerobus::zpz<509>; using type
05285
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<507»; }; // NOLINT
05286
           template<> struct ConwayPolynomial<509, 8> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<473>, ZPZV<482>, ZPZV<482>, ZPZV<473>, ZPZV<382>, ZPZV<2»; }; //
       NOLINT
05287
           template<> struct ConwayPolynomial<509, 9> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<314>, ZPZV<28>, ZPZV<507»;
       }; // NOLINT
           template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
05288
       POLYV<ZPZV<1>, ZPZV<518»; }; // NOLINT
            template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
05289
       POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
05290
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518»; }; // NOLINT
05291
            template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<509>, ZPZV<5%; }; // NOLINT template<> struct ConwayPolynomial<521, 5> { using ZPZ = aerobus::zpz<521>; using type =
05292
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, 6> { using ZPZ = aerobus::zpz<521>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, 6> { using ZPZ = aerobus::zpz<521>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<315>, ZPZV<280>, ZPZV<3»; }; // NOLINT
05293
            template<> struct ConwayPolynomial<521, 7> { using ZPZ = aerobus::zpz<521>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<518»; };
05295
           template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<462>, ZPZV<407>, ZPZV<312>, ZPZV<3»; }; //
       NOLINT
           template<> struct ConwayPolynomial<521, 9> { using ZPZ = aerobus::zpz<521>; using type =
05296
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<181>, ZPZV<483>, ZPZV<518»;
       }; // NOLINT
05297
            template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
       POLYY<ZPZY<1>, ZPZY<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
05298
       POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial523, 3> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
05300
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
05301
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<521»; }; // NOLINT
05302
            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
05303
           template<> struct ConwayPolynomial<523, 7> { using ZPZ = aerobus::zpz<523>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 8> { using ZPZ = aerobus::zpz<523>; using type =
05304
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<518>, ZPZV<184>, ZPZV<380>, ZPZV<2»; }; //
       NOLINT
            template<> struct ConwayPolynomial<523, 9> { using ZPZ = aerobus::zpz<523>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>>, ZPZV<145>, ZPZV<145>, ZPZV<521»;
       }; // NOLINT
05306
           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 =
05307
       POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
05308
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539»; }; // NOLINT template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
05309
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
05310
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<39»; }; // NOLINT
            template<> struct ConwayPolynomial<541, 6> { using ZPZ = aerobus::zpz<541>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<239>, ZPZV<320>, ZPZV<69>, ZPZV<2»; }; // NOLINT
      template<> struct ConwayPolynomial<541, 7> { using ZPZ = aerobus::zpz<541>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<539»; }; // NOLINT
template<> struct ConwayPolynomial<541, 8> { using ZPZ = aerobus::zpz<541>; using type =
05312
05313
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<376>, ZPZV<108>, ZPZV<113>, ZPZV<2*; }; //
05314
           template<> struct ConwayPolynomial<541, 9> { using ZPZ = aerobus::zpz<541>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<340>, ZPZV<318>, ZPZV<539»;
       }; // NOLINT
05315
           template<> struct ConwavPolynomial<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 =
05316
       POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; }; // NOLINT
05317
           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 =
05318
      POLYV<ZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<334>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<547, 5> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<545»; }; // NOLINT
      template<> struct ConwayPolynomial<547, 6> { using ZPZ = aerobus::zpz<547>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<547, 7> { using ZPZ = aerobus::zpz<547>; using type =
05320
05321
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<545»; }; // NOLINT
           template<> struct ConwayPolynomial<547, 8> { using ZPZ = aerobus::zpz<547>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2»; };
       NOLINT
05323
           template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<238>, ZPZV<263>, ZPZV<545»;
       }; // NOLINT
05324
            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 =
05325
      POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
05326
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<3>, ZPZV<355»; }; // NOLINT template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type =
05327
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; }; // NOLINT
          template<> struct ConwayPolynomial<557, 5> { using ZPZ = aerobus::zpz<557>; using type =
05328
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<555»; }; // NOLINT template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type =
05329
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<202, ZPZV<253>, ZPZV<253>, ZPZV<22*; }; // NOLINT template<> struct ConwayPolynomial<557, 7> { using ZPZ = aerobus::zpz<557>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<555»; };
           template<> struct ConwayPolynomial<557, 8> { using ZPZ = aerobus::zpz<557>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<384>, ZPZV<113>, ZPZV<2»; }; //
       NOLINT
           template<> struct ConwayPolynomial<557, 9> { using ZPZ = aerobus::zpz<557>; using type =
05332
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<456>, ZPZV<434>, ZPZV<555»;
       }; // NOLINT
05333
           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 =
05334
      POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
05335
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; }; // NOLINT template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
       \verb"POLYV<ZPZV<1>, \ \verb"ZPZV<0>, \ \verb"ZPZV<20>, \ \verb"ZPZV<399>, \ \verb"ZPZV<2»; \ \verb"}; \ \ // \ \verb"NOLINT" 
05337
           template<> struct ConwayPolynomial<563, 5> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<561»; }; // NOLINT
           template<> struct ConwayPolynomial<563, 6> { using ZPZ = aerobus::zpz<563>; using type =
05338
       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
05339
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<55, ZPZV<561»; }; // NOLINT template<> struct ConwayPolynomial<563, 8> { using ZPZ = aerobus::zpz<563>; using type =
05340
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<176>, ZPZV<509>, ZPZV<2»; }; //
       NOLINT
05341
           template<> struct ConwayPolynomial<563, 9> { using ZPZ = aerobus::zpz<563>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<15>, ZPZV<19>, ZPZV<561»; };
       // NOLINT
05342
           template<> struct ConwayPolynomial<569, 1> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<566»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 2> { using ZPZ = aerobus::zpz<569>; using type =
05343
      POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
05344
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
05345
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3%; }; // NOLINT template<> struct ConwayPolynomial<569, 5> { using ZPZ = aerobus::zpz<569>; using type =
05346
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 6> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<50>, ZPZV<263>, ZPZV<480>, ZPZV<3»; }; // NOLINT
05348
           template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<566»; }; // NOLINT
05349
           template<> struct ConwayPolynomial<569, 8> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<527>, ZPZV<173>, ZPZV<241>, ZPZV<241>, ZPZV<3»; }; //
      template<> struct ConwayPolynomial<569, 9> { using ZPZ = aerobus::zpz<569>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<478>, ZPZV<478>, ZPZV<566>, ZPZV<566>;
       }; // NOLINT
           template<> struct ConwayPolynomial<571, 1> { using ZPZ = aerobus::zpz<571>; using type =
05351
       POLYV<ZPZV<1>, ZPZV<568»; }; // NOLINT
```

```
05352
            template<> struct ConwayPolynomial<571, 2> { using ZPZ = aerobus::zpz<571>; using type =
       POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
05353
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<568»; }; // NOLINT
template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3»; }; // NOLINT
05354
            template<> struct ConwayPolynomial<571, 5> { using ZPZ = aerobus::zpz<571>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<568»; }; // NOLINT
           template<> struct ConwayPolynomial<571, 6> { using ZPZ = aerobus::zpz<571>; using type =
05356
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<295>, ZPZV<33>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<571, 7> { using ZPZ = aerobus::zpz<571>; using type =
05357
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<568»; }; // NOLINT
            template<> struct ConwayPolynomial<571, 8> { using ZPZ = aerobus::zpz<571>; using type =
05358
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<363>, ZPZV<119>, ZPZV<371>, ZPZV<37»; }; //
       NOLINT
05359
           template<> struct ConwayPolynomial<571, 9> { using ZPZ = aerobus::zpz<571>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<545>, ZPZV<179>, ZPZV<568»;
       }; // NOLINT
05360
            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 =
       POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
05362
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572»; }; // NOLINT template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
05363
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<577, 5> { using ZPZ = aerobus::zpz<577>; using type =
05364
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<572»; }; // NOLINT template<> struct ConwayPolynomial<577, 6> { using ZPZ = aerobus::zpz<577>; using type =
05365
       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 struct convayPolynomial<577, 7> { using ZPZ = aerobus::zpz<577>; using type struct convayPolynomial<577, 7> { using ZPZ = aerobus::zpz<577>; using type struct convayPolynomial<577, 7> { using ZPZ = aerobus::zpz<577>; using type struct convayPolynomial<577.
05366
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<8>, ZPZV<572»; };
            template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type =
05367
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<32; }; //
       NOLINT
05368
            template<> struct ConwayPolynomial<577, 9> { using ZPZ = aerobus::zpz<577>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<576>, ZPZV<444>, ZPZV<572»;
       }; // NOLINT
05369
            template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
       POLYV<ZPZV<1>, ZPZV<585»; }; // NOLINT
05370
           template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
       POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<587, 3> { using ZPZ = aerobus::zpz<587>; using type =
05371
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585»; }; // NOLINT template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<444>, ZPZV<2»; }; // NOLINT
05373
           template<> struct ConwayPolynomial<587, 5> { using ZPZ = aerobus::zpz<587>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<585»; }; // NOLINT
05374
            template<> struct ConwayPolynomial<587, 6> { using ZPZ = aerobus::zpz<587>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<204>, ZPZV<121>, ZPZV<226>, ZPZV<2»; }; // NOLINT
05375
            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<5, ZPZV<3>, ZPZV<585»; }; // NoLII template<> struct ConwayPolynomial<587, 8> { using ZPZ = aerobus::zpz<587>; using type :
05376
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<492>, ZPZV<444>, ZPZV<91>, ZPZV<2»; };
       NOLINT
05377
       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>
       }; // NOLINT
            template<> struct ConwayPolynomial<593, 1> { using ZPZ = aerobus::zpz<593>; using type =
05378
       POLYV<ZPZV<1>, ZPZV<590»; }; // NOLINT
           template<> struct ConwayPolynomial<593, 2> { using ZPZ = aerobus::zpz<593>; using type =
05379
       POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3»; }; // NOLINT
05380
            template<> struct ConwayPolynomial<593, 3> { using ZPZ = aerobus::zpz<593>; using type =
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<6>, ZPZY<590»; }; // NOLINT template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
05381
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<593, 5> { using ZPZ = aerobus::zpz<593>; using type =
05382
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<590»; }; // NOLINT
           template<> struct ConwayPolynomial<593, 6> { using ZPZ = aerobus::zpz<593>; using type =
05383
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<345>, ZPZV<65>, ZPZV<478>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<593, 7> { using ZPZ = aerobus::zpz<593>; using type =
05384
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<590»; };
05385
           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»; }; //
       NOLINT
            template<> struct ConwayPolynomial<593, 9> { using ZPZ = aerobus::zpz<593>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223>, ZPZV<523>, ZPZV<590»;
       }; // NOLINT
05387
            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 =
05388
       POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<592»; };
                                                                // NOLINT
           template<> struct ConwayPolynomial<599, 4> { using ZPZ = aerobus::zpz<599>; using type =
05390
      POLYV<ZPZV<1>, ZPZV<3>, ZPZV<419>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<599, 5> { using ZPZ = aerobus::zpz<599>; using type =
05391
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<592»; };
           template<> struct ConwayPolynomial<599, 6> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7»; }; // NOLINT
05393
           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<2>, ZPZV<592»; }; // NOLINT template<> struct ConwayPolynomial<599, 8> { using ZPZ = aerobus::zpz<599>; using type =
05394
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<440>, ZPZV<37>, ZPZV<124>, ZPZV<124>, ZPZV<7»; };
           template<> struct ConwayPolynomial<599, 9> { using ZPZ = aerobus::zpz<599>; using type =
05395
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3+, ZPZV<98>, ZPZV<592»;
       }; // NOLINT
           template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type =
05396
      POLYV<ZPZV<1>, ZPZV<594»; }; // NOLINT
           template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
05398
           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 =
05399
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<601, 5> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<594»; }; // NOLINT
05401
           template<> struct ConwayPolynomial<601, 6> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<440>, ZPZV<49>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<601, 7> { using ZPZ = aerobus::zpz<601>; using type
05402
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<594»; };
           template<> struct ConwayPolynomial<601, 8> { using ZPZ = aerobus::zpz<601>, using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<241>, ZPZV<490>, ZPZV<47»; }; //
       NOLINT
      template<> struct ConwayPolynomial<601, 9> { using ZPZ = aerobus::zpz<601>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<487>, ZPZV<487>, ZPZV<590>, ZPZV<594»;</pre>
05404
       }; // NOLINT
05405
           template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
       POLYV<ZPZV<1>, ZPZV<604»; }; // NOLINT
          template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
05406
      POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
05407
      POLYY<ZPZY<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 =
05409
      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 =
05410
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<45>, ZPZV<478>, ZPZV<478>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type
05411
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<604»; }; // NOLINT
05412
           template<> struct ConwayPolynomial<607, 8> { using ZPZ = aerobus::zpz<607>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<468>, ZPZV<35>, ZPZV<449>, ZPZV<3»; };
       NOLINT
           template<> struct ConwayPolynomial<607, 9> { using ZPZ = aerobus::zpz<607>; using type =
05413
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<444>, ZPZV<129>, ZPZV<604»;
       }; // NOLINT
           template<> struct ConwayPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
05414
      POLYY<ZPZY<1>, ZPZY<611»; }; // NOLINT template<> struct ConwayPolynomial<613, 2> { using ZPZ = aerobus::zpz<613>; using type =
05415
      POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
05416
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<611»; }; // NOLINT
           template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
05417
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
05418
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0, ZPZV<0, ZPZV<0, ZPZV<61)*; }; // NOLINT template<> struct ConwayPolynomial<613, 6> { using ZPZ = aerobus::zpz<613>; using type =
05419
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<609>, ZPZV<595>, ZPZV<601>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<61:»; };
05421
          template<> struct ConwayPolynomial<613, 8> { using ZPZ = aerobus::zpz<613>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<489>, ZPZV<57>, ZPZV<539>, ZPZV<2»; }; //
       NOLINT
           template<> struct ConwayPolynomial<613, 9> { using ZPZ = aerobus::zpz<613>; using type =
05422
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<513>, ZPZV<516, ZPZV<611»;
       }; // NOLINT
05423
           template<> struct ConwayPolynomial<617, 1> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<614»; }; // NOLINT
           template<> struct ConwayPolynomial<617, 2> { using ZPZ = aerobus::zpz<617>; using type =
05424
      POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<614»; }; // NOLINT template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type =
05426
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<617, 5> { using ZPZ = aerobus::zpz<617>; using type =
05427
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<614»; }; // NOLINT
           template<> struct ConwayPolynomial<617, 6> { using ZPZ = aerobus::zpz<617>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<318>, ZPZV<595>, ZPZV<310>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<617, 7> { using ZPZ = aerobus::zpz<617>; using type =
05429
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<614»; }; // NOLINT template<> struct ConwayPolynomial<617, 8> { using ZPZ = aerobus::zpz<617>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<519>, ZPZV<501>, ZPZV<55>, ZPZV<3»; }; //
05430
```

```
NOLINT
         template<> struct ConwayPolynomial<617, 9> { using ZPZ = aerobus::zpz<617>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<388>, ZPZV<543>, ZPZV<614»;
         }; // NOLINT
05432
               template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<617»; }; // NOLINT
               template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
         POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
05434
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617»; }; // NOLINT template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
05435
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<619, 5> { using ZPZ = aerobus::zpz<619>; using type =
05436
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<617»; }; // NOLINT
05437
               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 =
05438
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<416>, ZPZV<383>, ZPZV<225>, ZPZV<2*; }; //
05440
               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
05441
               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 =
05442
         POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3»; }; // NOLINT
05443
               template<> struct ConwayPolynomial<631, 3> { using ZPZ = aerobus::zpz<631>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT template<> struct ConwayPolynomial<631, 4> { using ZPZ = aerobus::zpz<631>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<3»; }; // NOLINT
05444
               template<> struct ConwayPolynomial<631, 5> { using ZPZ = aerobus::zpz<631>; using type =
05445
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
05446
               template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<516>, ZPZV<541>, ZPZV<106>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<631, 7> { using ZPZ = aerobus::zpz<631>; using type
05447
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<628»; }; //
               template<> struct ConwayPolynomial<631, 8> { using ZPZ = aerobus::zpz<631>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<3»; }; //
        template<> struct ConwayPolynomial<631, 9> { using ZPZ = aerobus::zpz<631>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<296>, ZPZV<413>, ZPZV<413>, ZPZV<413>, ZPZV<413</pre>
05449
         }; // NOLINT
   template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
         POLYV<ZPZV<1>, ZPZV<638»; }; // NOLINT
05451
               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 =
05452
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<638»; }; // NOLINT template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type =
05453
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<629>, ZPZV<3»; }; // NOLINT
05454
               template<> struct ConwayPolynomial<641, 5> { using ZPZ = aerobus::zpz<641>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<638»; }; // NOLINT template<> struct ConwayPolynomial<641, 6> { using ZPZ = aerobus::zpz<641>; using type =
05455
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<294>, ZPZV<294>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<638»; };
               template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<3»; }; //
         NOLINT
         template<> struct ConwayPolynomial<641, 9> { using ZPZ = aerobus::zpz<641>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<141>, ZPZV<638»;
05458
         }; // NOLINT
               template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
05459
         POLYV<ZPZV<1>, ZPZV<632»; }; // NOLINT
05460
               template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
         POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11»; }; // NOLINT
               template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
05461
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<632»; }; // NOLINT template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
05462
          \verb"POLYV<ZPZV<1>, \ \verb"ZPZV<0>, \ \verb"ZPZV<600>, \ \verb"ZPZV<11"; \ \verb"}; \ \ // \ \verb"NOLINT" 
               template<> struct ConwayPolynomial<643, 5> { using ZPZ = aerobus::zpz<643>; using type =
05463
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<632»; }; // NOLINT
               template<> struct ConwayPolynomial<643, 6> { using ZPZ = aerobus::zpz<643>; using type =
05464
         POLYV<2PZV<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
05465
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<632»; }; // NOLINT template<> struct ConwayPolynomial<643, 8> { using ZPZ = aerobus::zpz<643>; using type =
05466
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<631>, ZPZV<573>, ZPZV<569>, ZPZV<11»; }; //
         NOLINT
05467
               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<0>, ZPZV<591>, ZPZV<591>, ZPZV<4755, ZPZV<632»;
         }; // NOLINT
05468
               POLYV<ZPZV<1>, ZPZV<642»; }; // NOLINT
               template<> struct ConwayPolynomial<647, 2> { using ZPZ = aerobus::zpz<647>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5»; };
               template<> struct ConwayPolynomial<647, 3> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<642»; }; // NOLINT template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
05471
                                                                                               // NOLINT
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<647, 5> { using ZPZ = aerobus::zpz<647>; using type =
05472
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<642»; }; // NOLINT
               template<> struct ConwayPolynomial<647, 6> { using ZPZ = aerobus::zpz<647>; using type =
05473
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<308>, ZPZV<385>, ZPZV<642>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<647, 7> { using ZPZ = aerobus::zpz<647>; using type =
05474
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<642»; }; // NOLINT 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»; }; //
05476
              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
05477
               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 =
         POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2»; }; // NOLINT
05479
               template<> struct ConwayPolynomial<653, 3> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<651»; }; // NOLINT
template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2»; }; // NOLINT
05480
               template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651»; }; // NOLINT
05482
               template<> struct ConwayPolynomial<653, 6> { using ZPZ = aerobus::zpz<653>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<45>, ZPZV<220>, ZPZV<242>, ZPZV<2*; }; // NOLINT
05483
              template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type =
        POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51»; }; // NOLINT template<> struct ConwayPolynomial<653, 8> { using ZPZ = aerobus::zpz<653>; using type =
05484
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<385>, ZPZV<18>, ZPZV<296>, ZPZV<22»; };
         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<65), ZPZV<65), ZPZV<651»;</pre>
05485
         }; // NOLINT
               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 =
        POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
05488
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<657»; }; // NOLINT template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
05489
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2»; }; // NOLINT
05490
               template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<657»; }; // NOLINT
05491
              template<> struct ConwayPolynomial<659, 6> { using ZPZ = aerobus::zpz<659>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<371>, ZPZV<105>, ZPZV<223>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<659, 7> { using ZPZ = aerobus::zpz<659>; using type =
05492
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<657»; };
               template<> struct ConwayPolynomial<659, 8> { using ZPZ = aerobus::zpz<659>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<358>, ZPZV<246>, ZPZV<90>, ZPZV<2»; }; //
              template<> struct ConwayPolynomial<659, 9> { using ZPZ = aerobus::zpz<659>; using type =
05494
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<592>, ZPZV<592>, ZPZV<46>, ZPZV<657»;
05495
              template<> struct ConwayPolynomial<661, 1> { using ZPZ = aerobus::zpz<661>; using type =
         POLYV<ZPZV<1>, ZPZV<659»; }; // NOLINT
               template<> struct ConwayPolynomial<661, 2> { using ZPZ = aerobus::zpz<661>; using type =
05496
         POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
05497
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<659»; }; // NOLINT
               template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2»; }; // NOLINT
05499
              template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
          \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<19>, \verb"ZPZV<659"; \verb"}; $ // \verb"NOLINT" | NOLINT" | NOLINT"
        template<> struct ConwayPolynomial<661, 6> { using ZPZ = aerobus::zpz<661>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<456>, ZPZV<382>, ZPZV<2»; }; // NOLINT
05500
05501
               template<> struct ConwayPolynomial<661,
                                                                           7> { using ZPZ = aerobus::zpz<661>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<659»; };
              template<> struct ConwayPolynomial<661, 8> { using ZPZ = aerobus::zpz<661>; using type =
05502
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<612>, ZPZV<285>, ZPZV<72>, ZPZV<2»; }; //
         NOLINT
              template<> struct ConwayPolynomial<661, 9> { using ZPZ = aerobus::zpz<661>; using type =
05503
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<389>, ZPZV<220>, ZPZV<659»;
05504
               template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
         POLYV<ZPZV<1>, ZPZV<668»; }; // NOLINT
               template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
05505
         POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5»; }; // NOLINT
               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 =
05507
        POLYV<ZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<416>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<673, 5> { using ZPZ = aerobus::zpz<673>; using type =
05508
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<668»; }; // NOLINT
```

```
05509
                template<> struct ConwayPolynomial<673, 6> { using ZPZ = aerobus::zpz<673>; using type =
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<524>, ZPZV<248>, ZPZV<35>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<673, 7> { using ZPZ = aerobus::zpz<673>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<66, ZPZV<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 8> { using ZPZ = aerobus::zpz<673>; using type =
0.5.511
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<669>, ZPZV<587>, ZPZV<302>, ZPZV<5»; }; //
         NOLINT
                template<> struct ConwayPolynomial<673, 9> { using ZPZ = aerobus::zpz<673>; using type
05512
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<347>, ZPZV<553>, ZPZV<668»;
         }; // NOLINT
05513
               template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
         POLYV<ZPZV<1>, ZPZV<675»; }; // NOLINT
               template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
05514
         POLYV<ZPZV<1>, ZPZV<672>, ZPZV<2»; }; // NOLINT
05515
               template<> struct ConwayPolynomial<677, 3> { using ZPZ = aerobus::zpz<677>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<675»; }; // NOLINT template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
05516
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<631}, ZPZV<21>; // NOLINT template<> struct ConwayPolynomial<677, 5> { using ZPZ = aerobus::zpz<677>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<675»; }; // NOLINT
                template<> struct ConwayPolynomial<677, 6> { using ZPZ = aerobus::zpz<677>; using type =
05518
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446>, ZPZV<632>, ZPZV<50>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type =
05519
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<6>, ZPZV<6 , ZPZV<6 
05520
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361>, ZPZV<619>, ZPZV<6152>, 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<504>, ZPZV<504>, ZPZV<404>, ZPZV<675»;
05521
         }; // NOLINT
                template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
05522
         POLYV<ZPZV<1>, ZPZV<678»; }; // NOLINT
               template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
05523
         POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; }; // NOLINT
05524
                template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<678»; }; // NOLINT template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5»; }; // NOLINT
05525
05526
                template<> struct ConwayPolynomial<683, 5> { using ZPZ = aerobus::zpz<683>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<678»; }; // NOLINT
         template<> struct ConwayPolynomial<683, 6> { using ZPZ = aerobus::zpz<683>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<644>, ZPZV<109>, ZPZV<434>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<683, 7> { using ZPZ = aerobus::zpz<683>; using type =
05527
05528
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<678»; };
               template<> struct ConwayPolynomial<683, 8> { using ZPZ = aerobus::zpz<683>; using type
         POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<383>, 2PZV<1844, 2PZV<65>, 2PZV<5»; };
         template<> struct ConwayPolynomial<683, 9> { using ZPZ = aerobus::zpz<683>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<678»;</pre>
05530
         }; // NOLINT
05531
                template<> struct ConwayPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
         POLYV<ZPZV<1>, ZPZV<688»; }; // NOLINT
05532
                template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
         POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
05533
         POLYY<ZPZY<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 =
05535
         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 =
05536
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<262>, ZPZV<262>, ZPZV<262>; ; // NOLINT template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type
05537
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; }; //
05538
               template<> struct ConwayPolynomial<691, 8> { using ZPZ = aerobus::zpz<691>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<356>, ZPZV<425>, ZPZV<321>, ZPZV<3»; }; //
         NOLINT
05539
               template<> struct ConwayPolynomial<691, 9> { using ZPZ = aerobus::zpz<691>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<55, ZPZV<556>, ZPZV<443>, ZPZV<688»;
         }; // NOLINT
               template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
05540
         POLYV<ZPZV<1>, ZPZV<699»; }; // NOLINT
                template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
05541
         POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
05542
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; }; // NOLINT
               template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
05543
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
05544
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699»; }; // NOLINT
         template<> struct ConwayPolynomial</ri>
rollyv<zpzv<1>, ZPZv<3>, ZPZv<3>, ZPZv<3>, ZPZv<3>); // NoLINT
rollyv<Zpzv<1>, ZPZv<0>, ZPZv<1>, ZPZv<5>1>, ZPZv<32>, ZPZv<285>, ZPZv<2»; }; // NOLINT</pre>
05545
                template<> struct ConwayPolynomial<701, 7> { using ZPZ = aerobus::zpz<701>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<699»; };
05547
              template<> struct ConwayPolynomial<701, 8> { using ZPZ = aerobus::zpz<701>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<619>, ZPZV<2062, ZPZV<593>, ZPZV<29; }; //
         NOLTNT
```

```
template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<459>, ZPZV<373>, ZPZV<699»;
      }; // NOLINT
05549
           template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<707»; }; // NOLINT
           template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
05550
      POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
05551
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707»; }; // NOLINT template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
05552
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; }; // NOLINT
templated struct ConwayPolynomial<709, 5> { using ZPZ = aerobus::zpz<709>; using type =
05553
      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»; };
05555
           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<4>, ZPZV<707»; }; // NOLINT template<> struct ConwayPolynomial<709, 8> { using ZPZ = aerobus::zpz<709>; using type =
05556
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<689>, ZPZV<233>, ZPZV<79>, ZPZV<29; };
      template<> struct ConwayPolynomial<709, 9> { using ZPZ = aerobus::zpz<709>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<257>, ZPZV<2171>, ZPZV<707»;
05557
      }; // NOLINT
           template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
05558
      POLYV<ZPZV<1>, ZPZV<708»; }; // NOLINT
           template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type =
05560
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
05561
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<602>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<719, 5> { using ZPZ = aerobus::zpz<719>; using type =
05562
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
          template<> struct ConwayPolynomial<719, 6> { using ZPZ = aerobus::zpz<719>; using type =
05563
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<533>, ZPZV<591>, ZPZV<182>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<719, 7> { using ZPZ = aerobus::zpz<719>; using type =
05564
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT template<> struct ConwayPolynomial<719, 8> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<714>, ZPZV<362>, ZPZV<244>, ZPZV<11»; };
05566
          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
05567
           template<> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<722»; }; // NOLINT
05568
           template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; }; // NOLINT
          template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
05569
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<722»; }; // NOLINT
           template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
05570
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<727, 5> { using ZPZ = aerobus::zpz<727>; using type =
      05572
           template<> struct ConwayPolynomial<727, 6> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<86>, ZPZV<397>, ZPZV<672>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<727, 7> { using ZPZ = aerobus::zpz<727>; using type
05573
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<12»; }; // NOLINT
          template<> struct ConwayPolynomial<727, 8> { using ZPZ = aerobus::zpz<727>; using type =
05574
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<63>, ZPZV<639>, ZPZV<671>, ZPZV<368>, ZPZV<5»; };
      NOLINT
05575
          template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<573>, ZPZV<502>, ZPZV<722»;
      }; // NOLINT
           template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<727»; }; // NOLINT
05577
           template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; }; // NOLINT
           template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
05578
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727»; }; // NOLINT
           template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
05579
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<6»; }; // NOLINT
05580
          template<> struct ConwayPolynomial<733, 5> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<727»; }; // NOLINT template<> struct ConwayPolynomial<733, 6> { using ZPZ = aerobus::zpz<733>; using type =
05581
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, 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<0>, ZPZV<3>, ZPZV<3>, ZPZV<727»; };
05583
          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 =
05584
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<337>, ZPZV<6>, ZPZV<727»; };
05585
          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 =
05586
      POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; }; // NOLINT
```

```
05587
               template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; }; // NOLINT template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
05588
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
05589
        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 =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<447>, ZPZV<625>, ZPZV<3»; }; // NOLINI
              template<> struct ConwayPolynomial<739, 7> { using ZPZ = aerobus::zpz<739>; using type
05591
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<736»; };
05592
              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»: };
               template<> struct ConwayPolynomial<739, 9> { using ZPZ = aerobus::zpz<739>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<616>, ZPZV<81>, ZPZV<83>,
         }; // NOLINT
              template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
05594
        POLYV<ZPZV<1>, ZPZV<738»; }; // NOLINT
               template<> struct ConwayPolynomial<743, 2> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<743, 3> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<738»; }; // NOLINT template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
05597
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<743, 5> { using ZPZ = aerobus::zpz<743>; using type =
05598
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT
              template<> struct ConwayPolynomial<743, 6> { using ZPZ = aerobus::zpz<743>; using type =
05599
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<236>, ZPZV<471>, ZPZV<88>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<743, 7> { using ZPZ = aerobus::zpz<743>; using type :
05600
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<738»; }; // NOLINT
              template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
05601
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<279>, ZPZV<588>, ZPZV<58; }; //
              template<> struct ConwayPolynomial<743, 9> { using ZPZ = aerobus::zpz<743>; using type =
05602
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<327>, ZPZV<676>, ZPZV<738»;
         }; // NOLINT
        template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type = POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
05603
               template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
         POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT
05605
              template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; }; // NOLINT template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<525>, ZPZV<3»; }; // NOLINT
05606
               template<> struct ConwayPolynomial<751, 5> { using ZPZ = aerobus::zpz<751>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT
05608
              template<> struct ConwayPolynomial<751, 6> { using ZPZ = aerobus::zpz<751>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<298>, ZPZV<633>, ZPZV<539>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<751, 7> { using ZPZ = aerobus::zpz<751>; using type =
05609
        POLYV<ZPZV<1>, 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
05610
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<741>, ZPZV<243>, ZPZV<672>, ZPZV<3»; };
         NOLINT
        \label{eq:convergence} $$ \text{template}<> \text{struct ConwayPolynomial}<751, 9> $ \{ \text{using ZPZ} = \frac{\text{aerobus}::zpz}<751>; \text{using type} = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<703>, ZPZV<489>, 
05611
         }; // NOLINT
05612
               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 =
05613
        POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
05614
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
05615
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<757, 5> { using ZPZ = aerobus::zpz<757>; using type =
05616
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<755»; }; // NOLINT
05617
               template<> struct ConwayPolynomial<757, 6> { using ZPZ = aerobus::zpz<757>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<753>, ZPZV<745>, ZPZV<745>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<757, 7> { using ZPZ = aerobus::zpz<755>; using type
05618
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<755»; }; //
              template<> struct ConwayPolynomial<757, 8> { using ZPZ = aerobus::zpz<757>; using type 
05619
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<110>, ZPZV<509>, ZPZV<2»; }; //
         NOLINT
05620
              template<> struct ConwayPolynomial<757, 9> { using ZPZ = aerobus::zpz<757>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<688>, ZPZV<702>, ZPZV<755»;
         }; // NOLINT
05621
               template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
         POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
              template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
05622
        POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; }; // NOLINT
              template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
05623
        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
05625
             template<> struct ConwayPolynomial<761, 5> { using ZPZ = aerobus::zpz<761>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT
05626
              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»; };
               template<> struct ConwayPolynomial<761, 7> { using ZPZ = aerobus::zpz<761>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<755»; };
05628
              template<> struct ConwayPolynomial<761, 8> { using ZPZ = aerobus::zpz<761>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<603>, ZPZV<144>, ZPZV<540>, ZPZV<6*, }; //
        NOLTNT
05629
               template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<571>, ZPZV<755»;
         }; // NOLINT
05630
               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 =
05631
        POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; };
                                                                     // NOLINT
               template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; }; // NOLINT
              \texttt{template<> struct ConwayPolynomial<769, 4> \{ using \ \texttt{ZPZ} = aerobus::zpz<769>; \ using \ \texttt{type} = aerobus::zpz<769>; \ using \ \texttt
05633
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<769, 5> { using ZPZ = aerobus::zpz<769>; using type =
05634
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<758»; }; // NOLINT
               template<> struct ConwayPolynomial<769, 6> { using ZPZ = aerobus::zpz<769>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<326>, ZPZV<650>, ZPZV<11»; }; // NOLIN
05636
               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<0>, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<758»; }; // NOLINT template<> struct ConwayPolynomial<769, 8> { using ZPZ = aerobus::zpz<769>; using type =
05637
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<560>, ZPZV<574>, ZPZV<632>, ZPZV<11»; }; //
              template<> struct ConwayPolynomial<769, 9> { using ZPZ = aerobus::zpz<769>; using type =
05638
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<62>, ZPZV<751>, ZPZV<758»;
         }; // NOLINT
05639
              template<> struct ConwayPolynomial<773, 1> { using ZPZ = aerobus::zpz<773>; using type =
        POLYV<ZPZV<1>, ZPZV<771»; }; // NOLINT
05640
               template<> struct ConwayPolynomial<773, 2> { using ZPZ = aerobus::zpz<773>; using type =
         POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
05641
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<771»; }; // NOLINT template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
05642
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<773, 5> { using ZPZ = aerobus::zpz<773>; using type =
        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 =
05644
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<91>, ZPZV<3>, ZPZV<581>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<773, 7> { using ZPZ = aerobus::zpz<773>; using type =
05645
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<71»; }; // NOLINT
              template<> struct ConwayPolynomial<773, 8> { using ZPZ = aerobus::zpz<773>; using type =
05646
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<484>, ZPZV<94>, ZPZV<693>, ZPZV<2»; };
         NOLINT
05647
              template<> struct ConwayPolynomial<773, 9> { using ZPZ = aerobus::zpz<773>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<216>, ZPZV<574>, ZPZV<771»;
         }; // NOLINT
05648
               template<> struct ConwavPolynomial<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 =
        POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
05650
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785»; }; // NOLINT template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type =
05651
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2»; };
                                                                                               // NOLINT
              template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
05652
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<785»; }; // NOLINT
05653
               template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<98>, ZPZV<512>, ZPZV<606>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<787, 7> { using ZPZ = aerobus::zpz<787>; using type
05654
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<785»; };
               template<> struct ConwayPolynomial<787, 8> { using ZPZ = aerobus::zpz<787>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<612>, ZPZV<26>, ZPZV<715>, ZPZV<2»; };
         NOLINT
05656
        template<> struct ConwayPolynomial<787, 9> { using ZPZ = aerobus::zpz<787>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<480>, ZPZV<573>, ZPZV<785»;</pre>
         }; // NOLINT
05657
               template<> struct ConwayPolynomial<797, 1> { using ZPZ = aerobus::zpz<797>; using type =
        POLYV<ZPZV<1>, ZPZV<795»; }; // NOLINT
05658
              template<> struct ConwayPolynomial<797, 2> { using ZPZ = aerobus::zpz<797>; using type =
        POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
05659
        POLYY<ZPZY<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
05661
               template<> struct ConwayPolynomial<797, 5> { using ZPZ = aerobus::zpz<797>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<795»; }; // NOLINT
05662
               template<> struct ConwayPolynomial<797, 6> { using ZPZ = aerobus::zpz<797>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<657>, ZPZV<396>, ZPZV<71>, ZPZV<2»; }; // NOLINT
05663
               template<> struct ConwayPolynomial<797,
                                                                          7> { using ZPZ = aerobus::zpz<797>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<795»; }; // NOL template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type :
05664
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2»; }; //
        NOLINT
05665
              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<29>, ZPZV<599>, ZPZV<795»;
         }; // NOLINT
05666
              template<> struct ConwayPolynomial<809, 1> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<806»; }; // NOLINT
               template<> struct ConwayPolynomial<809, 2> { using ZPZ = aerobus::zpz<809>; using type =
05667
        POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<806»; }; // NOLINT template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
05669
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<809, 5> { using ZPZ = aerobus::zpz<809>; using type =
05670
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; }; // NOLINT
05671
               template<> struct ConwayPolynomial<809, 6> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<562>, ZPZV<75>, ZPZV<43>, ZPZV<3»; }; // NOLINT
05672
              template<> struct ConwayPolynomial<809, 7> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<806»; }; // NOLINT template<> struct ConwayPolynomial<809, 8> { using ZPZ = aerobus::zpz<809>; using type =
05673
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<593>, ZPZV<745>, ZPZV<673>, ZPZV<3»; }; //
05674
              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
05675
              template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
        POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
05676
               template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
        POLYV<ZPZV<1>, ZPZV<806>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<811, 3> { using ZPZ = aerobus::zpz<811>; using type =
05677
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<808»; }; // NOLINT template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type =
05678
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<811, 5> { using ZPZ = aerobus::zpz<811>; using type =
05679
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<808»; }; // NOLINT
               template<> struct ConwayPolynomial<811, 6> { using ZPZ = aerobus::zpz<811>; using type =
05680
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<780>, ZPZV<755>, ZPZV<307>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<811, 7> { using ZPZ = aerobus::zpz<811>; using type =
0.5681
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<808»; }; // NOLINT
              template<> struct ConwayPolynomial<811, 8> { using ZPZ = aerobus::zpz<811>; using type =
05682
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<525>, ZPZV<3»; }; //
        template<> struct ConwayPolynomial<811, 9> { using ZPZ = aerobus::zpz<811>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<382>, ZPZV<200>, ZPZV<808»;
05683
         }; // NOLINT
               template<> struct ConwayPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
05684
        POLYV<ZPZV<1>, ZPZV<819»; }; // NOLINT
               template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT
05686
              template<> struct ConwayPolynomial<821, 3> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<819»; }; // NOLINT template<> struct ConwayPolynomial<821, 4> { using ZPZ = aerobus::zpz<821>; using type =
05687
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<15>, ZPZV<662>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
05688
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<819»; }; // NOLINT
05689
               template<> struct ConwayPolynomial<821, 6> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<160>, ZPZV<130>, ZPZV<803>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type =
05690
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, 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 =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<556>, ZPZV<589>, ZPZV<2»; }; //
05692
               template<> struct ConwayPolynomial<821, 9> { using ZPZ = aerobus::zpz<821>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<819»;
         }; // NOLINT
05693
               template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
05694
              template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; }; // NOLINT
05695
               template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<3
05696
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
05697
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
05698
               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 template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type
05699
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<820»; }; // NOLINT
              template<> struct ConwayPolynomial<823, 8> { using ZPZ = aerobus::zpz<823>; using type =
05700
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<451>, ZPZV<437>, ZPZV<31>, ZPZV<33; };
         NOLINT
        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»;</pre>
05701
        }; // NOLINT
    template<> struct ConwayPolynomial<827, 1> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<825»; }; // NOLINT
              template<> struct ConwayPolynomial<827, 2> { using ZPZ = aerobus::zpz<827>; using type =
05703
        POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825»; };
        template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<827, 5> { using ZPZ = aerobus::zpz<827>; using type =
05705
05706
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<825»; }; // NOLINT
        template<> struct ConwayPolynomial<827, 6> { using ZPZ = aerobus::zpz<827>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<685>, ZPZV<601>, ZPZV<691>, ZPZV<2»; }; // NOLINT
05707
05708
               template<> struct ConwayPolynomial<827, 7> { using ZPZ = aerobus::zpz<827>; using type
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<825»; }; // NOLINT template<> struct ConwayPolynomial<827, 8> { using ZPZ = aerobus::zpz<827>; using type =
05709
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<812>, ZPZV<79>, ZPZV<32>, ZPZV<2»; };
         NOLINT
05710
               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<1>7>, ZPZV<372>, ZPZV<825»;
         }; // NOLINT
0.5711
               template<> struct ConwayPolynomial<829, 1> { using ZPZ = aerobus::zpz<829>; using type =
         POLYV<ZPZV<1>, ZPZV<827»; }; // NOLINT
               template<> struct ConwayPolynomial<829, 2> { using ZPZ = aerobus::zpz<829>; using type =
05712
         POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<5
05714
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<604>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<829, 5> { using ZPZ = aerobus::zpz<829>; using type =
05715
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<827»; }; // NOLINT
               template<> struct ConwayPolynomial<829, 6> { using ZPZ = aerobus::zpz<829>; using type =
         POLYV<2PZV<1>, 2PZV<0>, 2PZV<1>, 2PZV<341>, 2PZV<476>, 2PZV<817>, 2PZV<2»; }; // NOLINI
05717
               template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<827»; }; // NOLINT
05718
              template<> struct ConwayPolynomial<829, 8> { using ZPZ = aerobus::zpz<829>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<468>, ZPZV<241>, ZPZV<138>, ZPZV<2*; }; //
         NOLINT
               template<> struct ConwayPolynomial<829, 9> { using ZPZ = aerobus::zpz<829>; using type =
05719
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<621>, ZPZV<552>, ZPZV<827»;
         }; // NOLINT
05720
               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 =
         POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; }; // NOLINT
              template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; }; // NOLINT template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
05723
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
05724
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<828»; }; // NOLINT
05725
               template<> struct ConwayPolynomial<839, 6> { using ZPZ = aerobus::zpz<839>; using type =
         05726
              template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<828»; }; // NOLINT
              template<> struct ConwayPolynomial<839, 8> { using ZPZ = aerobus::zpz<839>; using type
05727
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<553>, ZPZV<779>, ZPZV<329>, ZPZV<11»; }; //
05728
              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
05729
               template<> struct ConwayPolynomial<853, 1> { using ZPZ = aerobus::zpz<853>; using type =
         POLYV<ZPZV<1>, ZPZV<851»; }; // NOLINT
              template<> struct ConwayPolynomial<853, 2> { using ZPZ = aerobus::zpz<853>; using type =
         POLYV<ZPZV<1>, ZPZV<852>, ZPZV<2»; }; // NOLINT
05731
               template<> struct ConwayPolynomial<853, 3> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851»; ); // NOLINT
template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<2»; }; // NOLINT
05732
                template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<851»; // NOLINT
05734
              template<> struct ConwayPolynomial<853, 6> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<276>, ZPZV<194>, ZPZV<512>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type =
05735
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<851»; }; // NOLINT
               template<> struct ConwayPolynomial<853, 8> { using ZPZ = aerobus::zpz<853>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8+, ZPZV<846>, 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<67>, ZPZV<677>, ZPZV<821>, ZPZV<851»;</pre>
05737
        }; // NOLINT
    template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
         POLYV<ZPZV<1>, ZPZV<854»; }; // NOLINT
05739
              template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
        POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
05740
        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<5x; }; // NOLINT template<> struct ConwayPolynomial<857, 5> { using ZPZ = aerobus::zpz<857>; using type =
05742
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<854»; }; // NOLINT template<> struct ConwayPolynomial<857, 6> { using ZPZ = aerobus::zpz<857>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<824>, ZPZV<85>, ZPZV<3»; }; // NOLINT
05743
```

```
template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::zpz<857>; using type
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<854»; }; // NOLINT
                    template<> struct ConwayPolynomial<857, 8> { using ZPZ = aerobus::zpz<857>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<611>, ZPZV<552>, ZPZV<494>, ZPZV<49*, }; //
             NOLINT
                    template<> struct ConwayPolynomial<857, 9> { using ZPZ = aerobus::zpz<857>; using type =
05746
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<319>, ZPZV<719>, ZPZV<854»;
             }; // NOLINT
05747
                    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 =
05748
            POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT
05749
                     template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857»; }; // NOLINT
05750
                    template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<859, 5> { using ZPZ = aerobus::zpz<859>; using type =
05751
            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0, ZPZV<1>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 6> { using ZPZ = aerobus::zpz<859>; using type =
            POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<419>, ZPZV<646>, ZPZV<566>, ZPZV<2»; }; // NOLINT
                     template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpz<859>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<857»; }; // NOLINT
05754
                    template<> struct ConwayPolynomial<859, 8> { using ZPZ = aerobus::zpz<859>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<52>, ZPZV<446>, ZPZV<672>, ZPZV<672>, ZPZV<2*; }; //
            NOLINT
                     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
05756
                     template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
            POLYV<ZPZV<1>, ZPZV<858»; }; // NOLINT
                    template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
05757
            POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; }; // NOLINT
                     template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<858»; }; // NOLINT
                     template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
05759
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5»; }; // NOLINT
                     template<> struct ConwayPolynomial<863, 5> { using ZPZ = aerobus::zpz<863>; using type =
05760
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<858»; }; // NOLINT
05761
                     template<> struct ConwayPolynomial<863, 6> { using ZPZ = aerobus::zpz<863>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<300, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<863, 7> { using ZPZ = aerobus::zpz<863>; using type =
05762
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<858»; }; // NOLINT template<> struct ConwayPolynomial<863, 8> { using ZPZ = aerobus::zpz<863>; using type =
05763
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<765>, ZPZV<576>, ZPZV<849>, ZPZV<5»; }; //
05764
                     template<> struct ConwayPolynomial<863, 9> { using ZPZ = aerobus::zpz<863>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<381>, ZPZV<31>, ZPZV<1>, ZPZV<858»; };
             // NOLINT
                     template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
05765
            POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
                     template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
            POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; }; // NOLINT
05767
                    template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875»; ); // NOLINT
template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<877, 5> { using ZPZ = aerobus::zpz<877>; using type =
05768
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<875»; }; // NOLINT
05770
                     template<> struct ConwayPolynomial<877, 6> { using ZPZ = aerobus::zpz<877>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<400>, ZPZV<855>, ZPZV<2»; }; // NOLINT
05771
                    template<> struct ConwayPolynomial<877, 7> { using ZPZ = aerobus::zpz<877>; using type =
            POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<875%; }; // NOLINT
                     template<> struct ConwayPolynomial<877, 8> { using ZPZ = aerobus::zpz<877>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<767>, ZPZV<319>, ZPZV<347>, ZPZV<34>, ZPZV<34>, ZPZV<319>, ZPZV<347>, ZPZV<347</p>
            template<> struct ConwayPolynomial<877, 9> { using ZPZ = aerobus::zpz<877>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<770>, ZPZV<278>, ZPZV<875»;
05773
             }; // NOLINT
                     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 =
05775
            POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT
                     template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
05776
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<878»; }; // NOLINT
template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<447>, ZPZV<3»; }; // NOLINT
05777
                    template<> struct ConwayPolynomial<881, 5> { using ZPZ = aerobus::zpz<881>; using type =
05778
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<878»; }; // NOLINT
05779
                    template<> struct ConwayPolynomial<881, 6> { using ZPZ = aerobus::zpz<881>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<419>, ZPZV<23>, ZPZV<3»; // NOLINT template<> struct CommayPolynomial<881, 7> { using zrz = aerobus::zpz<881>; using type = DNIV(ZPZV) | ZPZV<0 | ZPZV | ZPZ
05780
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                    template<> struct ConwayPolynomial<881, 8> { using ZPZ = aerobus::zpz<881>; using type
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<635>, ZPZV<490>, ZPZV<561>, ZPZV<3»; }; //
            template<> struct ConwayPolynomial<881, 9> { using ZPZ = aerobus::zpz<881>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<587>, ZPZV<510>, ZPZV<878»;</pre>
```

```
}; // NOLINT
05783
            template<> struct ConwayPolynomial<883, 1> { using ZPZ = aerobus::zpz<883>; using type =
      POLYV<ZPZV<1>, ZPZV<881»; }; // NOLINT
           template<> struct ConwayPolynomial<883, 2> { using ZPZ = aerobus::zpz<883>; using type =
05784
      POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; }; // NOLINT
05785
           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 =
05786
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<883, 5> { using ZPZ = aerobus::zpz<883>; using type =
05787
      POLYV-ZPZV-1>, ZPZV-(>, ZPZV-(>, ZPZV-(>), ZPZV-14>, ZPZV-881»; }; // NOLINT template<> struct ConwayPolynomial<883, 6> { using ZPZ = aerobus::zpz<883>; using type =
05788
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<87>, ZPZV<875, ZPZV<871>, ZPZV<871>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<881»; }; // NOLINT
05790
           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<5>, ZPZV<360>, ZPZV<360>, ZPZV<557>, ZPZV<881»;
       }; // NOLINT
05792
           template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
      POLYV<ZPZV<1>, ZPZV<882»; }; // NOLINT
           template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
05793
      POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; }; // NOLINT
            template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<882»; }; // NOLINT
           template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
05795
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<887, 5> { using ZPZ = aerobus::zpz<887>; using type =
05796
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<882»; }; // NOLINT
05797
            template<> struct ConwayPolynomial<887, 6> { using ZPZ = aerobus::zpz<887>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<775>, ZPZV<341>, ZPZV<28>, ZPZV<5»; }; // NOLINI
05798
          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<0>, ZPZV<8>, ZPZV<8>, ZPZV<88, ZPZV<882»; }; // NOLINT template<> struct ConwayPolynomial<887, 8> { using ZPZ = aerobus::zpz<887>; using type =
05799
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<781>, ZPZV<381>, ZPZV<706>, ZPZV<5»; }; //
05800
            template<> struct ConwayPolynomial<887, 9> { using ZPZ = aerobus::zpz<887>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<727>, ZPZV<345>, ZPZV<882»;
       }; // NOLINT
0.5801
           template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
      POLYV<ZPZV<1>, ZPZV<905»; }; // NOLINT
05802
           template<> struct ConwayPolynomial<907, 2> { using ZPZ = aerobus::zpz<907>; using type =
      POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2»; }; // NOLINT
05803
           template<> struct ConwayPolynomial<907, 3> { using ZPZ = aerobus::zpz<907>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<905»; }; // NOLINT template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
05804
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
05805
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; }; // NOLINT
           template<> struct ConwayPolynomial<907, 6> { using ZPZ = aerobus::zpz<907>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<626>, ZPZV<752>, ZPZV<266>, ZPZV<2»; }; // NOLINT
05807
           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<905»; }; // NOLINT
           template<> struct ConwayPolynomial<907, 8> { using ZPZ = aerobus::zpz<907>; using type =
05808
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<584>, ZPZV<518>, ZPZV<811>, ZPZV<2»; }; //
      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<783>, ZPZV<57>, ZPZV<905»;
       }; // NOLINT
05810
            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 =
      POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; }; // NOLINT
           template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
05812
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; }; // NOLINT template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
05813
      POLYVCZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17>; }; // NOLINT template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
05814
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<894»; }; // NOLINT
05815
           template<> struct ConwayPolynomial<911, 6> { using ZPZ = aerobus::zpz<911>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<17>, ZPZV<683>, ZPZV<19>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
05816
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<894»; };
                                                                                                         // NOLINT
            template<> struct ConwayPolynomial<911, 8> { using ZPZ = aerobus::zpz<911>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<708>, ZPZV<590>, ZPZV<168>, ZPZV<17»; };
05818
           template<> struct ConwayPolynomial<911, 9> { using ZPZ = aerobus::zpz<911>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<679>, ZPZV<116>, ZPZV<894»;
       }; // NOLINT
            template<> struct ConwayPolynomial<919, 1> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<912»; }; // NOLINT
05820
           template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912»; }; // NOLINT
05821
```

```
05822
           template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<*; }; // NOLINT template<> struct ConwayPolynomial<919, 5> { using ZPZ = aerobus::zpz<919>; using type =
05823
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<912»; }; // NOLINT
      template<> struct ConwayPolynomial<919, 6> { using ZPZ = aerobus::zpz<919>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<312>, ZPZV<817>, ZPZV<113>, ZPZV<7»; }; // NOLINT
05824
           template<> struct ConwayPolynomial<919, 7> { using ZPZ = aerobus::zpz<919>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<912»; };
          template<> struct ConwayPolynomial<919, 8> { using ZPZ = aerobus::zpz<919>; using type =
05826
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<202>, ZPZV<504>, ZPZV<7»; }; //
       NOLINT
           template<> struct ConwayPolynomial<919, 9> { using ZPZ = aerobus::zpz<919>; using type =
05827
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<410>, ZPZV<623>, ZPZV<912»;
       }; // NOLINT
05828
           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 =
05829
      POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<926»; }; // NOLINT template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
05832
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<926»; }; // NOLINT
05833
           template<> struct ConwayPolynomial<929, 6> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<805>, ZPZV<92>, ZPZV<86>, ZPZV<3»; }; // NOLINT
05834
           template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<926»; }; // NOLINT template<> struct ConwayPolynomial<929, 8> { using ZPZ = aerobus::zpz<929>; using type =
05835
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699>, ZPZV<292>, ZPZV<5865, ZPZV<3»; }; //
       NOLINT
05836
           template<> struct ConwayPolynomial<929, 9> { using ZPZ = aerobus::zpz<929>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<481>, ZPZV<199>, ZPZV<926»;
       }; // NOLINT
05837
           template<> struct ConwayPolynomial<937, 1> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<932»; }; // NOLINT
           template<> struct ConwayPolynomial<937, 2> { using ZPZ = aerobus::zpz<937>; using type =
05838
      POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5»; }; // NOLINT
05839
           template<> struct ConwayPolynomial<937, 3> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<932»; }; // NOLINT template<> struct ConwayPolynomial<937, 4> { using ZPZ = aerobus::zpz<937>; using type =
05840
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<23>, ZPZV<585>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
0.5841
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<932»; }; // NOLINT
           template<> struct ConwayPolynomial<937, 6> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<794>, 2PZV<727>, 2PZV<934>, 2PZV<5»; }; // NOLINT
05843
          template<> struct ConwayPolynomial<937, 7> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<24>, ZPZV<932»; }; // NOLINT
           template<> struct ConwayPolynomial<937, 8> { using ZPZ = aerobus::zpz<937>; using type =
05844
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<26>, ZPZV<53>, ZPZV<53>; };
           template<> struct ConwayPolynomial<937, 9> { using ZPZ = aerobus::zpz<937>; using type =
05845
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<533>, ZPZV<483>, ZPZV<932»;
       }; // NOLINT
05846
           template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<939»; }; // NOLINT
           template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
05848
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3939; }; // NOLINT
template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<941, 5> { using ZPZ = aerobus::zpz<941>; using type =
05849
05850
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<939»; }; // NOLINT
05851
           template<> struct ConwayPolynomial<941, 6> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<459>, ZPZV<694>, ZPZV<538>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<941, 7> { using ZPZ = aerobus::zpz<941>; using type =
05852
      POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<939»; }; // NOLINT
           template<> struct ConwayPolynomial<941, 8> { using ZPZ = aerobus::zpz<941>; using type =
05853
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<675>, ZPZV<590>, ZPZV<2»; };
05854
          template<> struct ConwayPolynomial<941, 9> { using ZPZ = aerobus::zpz<941>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708>, ZPZV<197>, ZPZV<939»;
       }; // NOLINT
05855
           template<> struct ConwayPolynomial<947, 1> { using ZPZ = aerobus::zpz<947>; using type =
       POLYV<ZPZV<1>, ZPZV<945»; }; // NOLINT
           template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
05856
       POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
05857
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 4> { using ZPZ = aerobus::zpz<947>; using type =
05858
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2»; };
                                                                       // NOLINT
           template<> struct ConwayPolynomial<947, 5> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<945»; }; // NOLINT
05860
          template<> struct ConwayPolynomial<947, 6> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<880>, ZPZV<787>, ZPZV<95>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<947, 7> { using ZPZ = aerobus::zpz<947>; using type =
05861
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<945»; };
                  template<> struct ConwayPolynomial<947, 8> { using ZPZ = aerobus::zpz<947>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<845>, ZPZV<597>, ZPZV<581>, ZPZV<2»; }; //
           NOLINT
05863
                  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<0>, ZPZV<269>, ZPZV<808>, ZPZV<945»;
05864
                  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 =
05865
          POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
05866
          POLYY<ZPZY<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
05868
                  template<> struct ConwayPolynomial<953, 5> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<950»; }; // NOLINT
          template<> struct ConwayPolynomial<953, 6> { using ZPZ = aerobus::zpz<953>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<507>, ZPZV<829>, ZPZV<730>, ZPZV<3»; }; // NOLINT
05869
                  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
05871
                  template<> struct ConwayPolynomial<953, 8> { using ZPZ = aerobus::zpz<953>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<579>, ZPZV<658>, ZPZV<108>, ZPZV<3»; }; //
           NOLINT
05872
                  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<0>, ZPZV<1>, ZPZV<819>, ZPZV<316>, ZPZV<950»;
           }; // NOLINT
05873
                  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 =
05874
          POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5»; }; // NOLINT
05875
                  template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<62,; }; // NOLINT template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
05876
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<967, 5> { using ZPZ = aerobus::zpz<967>; using type =
05877
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<2>, ZPZY<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 6> { using ZPZ = aerobus::zpz<967>; using type =
          POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<948>, ZPZV<831>, ZPZV<5»; }; // NOLINT
                  template<> struct ConwayPolynomial<967, 7> { using ZPZ = aerobus::zpz<967>; using type =
05879
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 8> { using ZPZ = aerobus::zpz<967>; using type =
05880
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<840>, ZPZV<502>, ZPZV<136>, ZPZV<5»; }; //
                  template<> struct ConwayPolynomial<967, 9> { using ZPZ = aerobus::zpz<967>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<512>, ZPZV<783>, ZPZV<962»;
           }; // NOLINT
05882
                  template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<965»; }; // NOLINT
                  template<> struct ConwayPolynomial<971, 2> { using ZPZ = aerobus::zpz<971>; using type =
05883
          POLYV<ZPZV<1>, ZPZV<970>, ZPZV<6»; }; // NOLINT
                  template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<965»; }; // NOLINT template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
05885
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<527>, ZPZV<6»; }; // NOLINT
                  template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
05886
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<965»; }; // NOLINT
                 template<> struct ConwayPolynomial<971, 6> { using ZPZ = aerobus::zpz<971>; using type =
05887
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<970>, ZPZV<729>, ZPZV<718>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<971, 7> { using ZPZ = aerobus::zpz<971>; using type =
05888
          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 =
                                                                                                                                                                        // NOLINT
05889
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<725>, ZPZV<281>, ZPZV<206>, ZPZV<6»; }; //
05890
                 template<> struct ConwayPolynomial<971, 9> { using ZPZ = aerobus::zpz<971>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<473>, ZPZV<965»;
           }; // NOLINT
05891
                  \texttt{template<> struct ConwayPolynomial<977, 1> \{ using ZPZ = aerobus:: zpz<977>; using type = 200 t
          POLYV<ZPZV<1>, ZPZV<974»; }; // NOLINT
05892
                  template<> struct ConwayPolynomial<977, 2> { using ZPZ = aerobus::zpz<977>; using type =
           POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; }; // NOLINT
05893
                 template<> struct ConwayPolynomial<977, 3> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<974»; }; // NOLINT template<> struct ConwayPolynomial<977, 4> { using ZPZ = aerobus::zpz<977>; using type =
05894
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<800>, ZPZV<800>, ZPZV<800>, ZPZV<800>; ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<977, 5> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<974»; }; // NOLINT
          template<> struct ConwayPolynomial<977, 6> { using ZPZ = aerobus::zpz<977>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<729>, ZPZV<830>, ZPZV<753>, ZPZV<3»; }; // NOLINT
05896
05897
                  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<7>, ZPZV<74*; }; // NOLINT
                  template<> struct ConwayPolynomial<977, 8> { using ZPZ = aerobus::zpz<977>; using type =
05898
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<855>, ZPZV<807>, ZPZV<77>, ZPZV<3»; };
           NOLINT
05899
                 template<> struct ConwayPolynomial<977, 9> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<740>, ZPZV<974»;
           }; // NOLINT
```

```
template<> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
         POLYV<ZPZV<1>, ZPZV<978»; }; // NOLINT
05901
               template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
         POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; }; // NOLINT
05902
                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
05904
               template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<978»; }; // NOLINT
                template<> struct ConwayPolynomial<983, 6> { using ZPZ = aerobus::zpz<983>; using type =
05905
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<84>, ZPZV<26>, ZPZV<28, ZPZV<28, ZPZV<28, ZPZV<28, ZPZV<28, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type
05906
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZV<3
05907
               template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<53»; }; //
          NOLTNT
         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<87>, ZPZV<87>, ZPZV<87>, ZPZV<87>, ZPZV<87>, ZPZV<87
05908
                template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
05909
         POLYV<ZPZV<1>, ZPZV<985»; }; // NOLINT
               template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
05910
         POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; }; // NOLINT
                template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
05911
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985»; }; // NOLINT
                template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
05912
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6»; }; // NOLINT
                template<> struct ConwayPolynomial<991, 5> { using ZPZ = aerobus::zpz<991>; using type =
05913
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<985»; }; // NOLINT
         template<> struct ConwayPolynomial
691, 65 { using ZPZ = aerobus::zpz<991>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<637>, ZPZV<855>, ZPZV<278>, ZPZV<6»; }; // NOLINT</pre>
05914
05915
                template<> struct ConwayPolynomial<991, 7> { using ZPZ = aerobus::zpz<991>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<985»; }; // NOLINT
         template<> struct ConwayPolynomial<991, 8> { using ZPZ = aerobus::zpz<991>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<941>, ZPZV<786>, ZPZV<234>, ZPZV<6»; }; //</pre>
05916
                template<> struct ConwayPolynomial<991, 9> { using ZPZ = aerobus::zpz<991>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<466>, ZPZV<422>, ZPZV<985»;
          }; // NOLINT
05918
                template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
         POLYV<ZPZV<1>, ZPZV<990»; }; // NOLINT
                template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
05919
         POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; }; // NOLINT
                template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990»; };
                                                                                        // NOLINT
               template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
05921
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
05922
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<990»; }; // NOLINT
05923
                template<> struct ConwayPolynomial<997, 6> { using ZPZ = aerobus::zpz<997>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<981>, ZPZV<58>, ZPZV<260>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<997, 7> { using ZPZ = aerobus::zpz<997>; using type =
05924
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<990»; }; // NOLINT template<> struct ConwayPolynomial<997, 8> { using ZPZ = aerobus::zpz<997>; using type =
05925
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<934>, ZPZV<473>, ZPZV<241>, ZPZV<241>, ZPZV<7»; }; //
               template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type =
05926
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<39>, ZPZV<732>, ZPZV<616>, ZPZV<990»;
          }; // NOLINT
05927 #endif // DO_NOT_DOCUMENT
05928 } // namespace aerobus
05929 #endif // AEROBUS_CONWAY_IMPORTS
05931 #endif // __INC_AEROBUS__ // NOLINT
```

#### 9.4 src/examples.h File Reference

#### 9.5 examples.h

#### Go to the documentation of this file.

```
00001 #ifndef SRC_EXAMPLES_H_
00002 #define SRC_EXAMPLES_H_
00042 #endif // SRC_EXAMPLES_H_
```

## **Chapter 10**

# **Examples**

#### 10.1 examples/hermite.cpp

How to use aerobus::known\_polynomials::hermite\_phys polynomials

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

## 10.2 examples/custom\_taylor.cpp

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

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

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

template<size_t deg>
```

202 Examples

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

#### 10.3 examples/fp16.cu

How to leverage CUDA \_\_half and \_\_half2 16 bits floating points number using aerobus::i16 Warning : due to an NVIDIA bug (lack of constexpr operators), performance is not good

```
// TO compile with nvcc -03 -std=c++20 -arch=sm_90 fp16.cu
#include <cstdio>

#define WITH_CUDA_FP16
#include "../src/aerobus.h"

/*
change int_type to aerobus::i32 (or i64) and float_type to float (resp. double)
to see how good is the generated assembly compared to what nvcc generates for 16 bits
*/
using int_type = aerobus::i16;
using float_type = __half2;

using EXPM1 = aerobus::expml<int_type, 6>;

__device__ INLINED float_type f(float_type x) {
    return EXPM1::eval(x);
}

__global__ void run(size_t N, float_type* in, float_type* out) {
    for(size_t i = threadIdx.x + blockDim.x * blockIdx.x; i < N; i += blockDim.x * gridDim.x) {
        out[i] = f(f(f(f(f(in[i]))))));
    }
}
int main() {
}</pre>
```

### 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 <iostream>
#include "../src/aerobus.h"
// let's build Abel polynomials from scratch using Aerobus
// note : it's now integrated in the main library, but still serves as an example
template<typename I = aerobus::i64>
struct AbelHelper {
private:
    using P = aerobus::polynomial<I>;
 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>>
    };
    // \text{ abel}(0, a) = 1
    template<I::inner_type an>
    struct Inner<0, an>
        using type = P::one;
    // abel(1, a) = X
    template<I::inner_type an>
    struct Inner<1, an>
        using type = P::X;
    };
};
template<size_t n, auto a, typename I = aerobus::i64>
using AbelPolynomials = typename AbelHelper<I>::template Inner<n, a*n>::type;
using A2_3 = AbelPolynomials<3, 2>;
    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>
```

204 Examples

```
#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 w "expected = " w 1 w std::endl;
    std::cout w "computed = " w x w std::endl;
    return 0;
}
```

## Index

```
abs t
                                                              mulfractions t, 29
     aerobus, 20
                                                              pi64, 30
add t
                                                              PI fraction, 30
    aerobus, 20
                                                              pow t, 30
    aerobus::i32, 55
                                                              pq64, 30
    aerobus::i64, 61
                                                              q32, 30
    aerobus::polynomial < Ring >, 69
                                                              q64, 31
    aerobus::Quotient < Ring, X >, 77
                                                              sin, 31
    aerobus::zpz, 102
                                                              sinh, 31
                                                              SQRT2 fraction, 31
addfractions t
    aerobus, 20
                                                              SQRT3 fraction, 31
aerobus, 15
                                                              stirling_1_signed_t, 32
    abs_t, 20
                                                              stirling_1_unsigned_t, 32
    add_t, 20
                                                              stirling_2_t, 32
    addfractions t, 20
                                                              sub t, 33
    aligned_malloc, 34
                                                              tan, 33
    alternate_t, 20
                                                              tanh, 33
    alternate_v, 35
                                                              taylor, 33
    asin, 21
                                                              vadd t, 34
    asinh, 21
                                                              vmul_t, 34
                                                         aerobus::ContinuedFraction < a0 >, 45
    atan, 21
    atanh, 21
                                                              type, 45
    bell t, 23
                                                              val, 46
    bernoulli t, 23
                                                         aerobus::ContinuedFraction < a0, rest... >, 46
    bernoulli v, 35
                                                              type, 47
    combination t, 23
                                                              val, 47
    combination v, 35
                                                         aerobus::ContinuedFraction < values >, 44
                                                         aerobus::ConwayPolynomial, 47
    cos, 23
    cosh, 25
                                                         aerobus::Embed< i32, i64 >, 48
    div t, 25
                                                              type, 48
    E fraction, 25
                                                         aerobus::Embed< polynomial< Small >, polynomial<
    embed_int_poly_in_fractions_t, 25
                                                                  Large > >, 49
    exp, 26
                                                              type, 49
    expm1, 26
                                                         aerobus::Embed < q32, q64 >, 50
    factorial t, 26
                                                              type, 50
    factorial_v, 35
                                                         aerobus::Embed< Quotient< Ring, X >, Ring >, 50
    field, 34
    fpq32, 26
                                                         aerobus::Embed< Ring, FractionField< Ring >>, 51
    fpq64, 27
                                                              type, 52
                                                         aerobus::Embed< Small, Large, E >, 47
    FractionField, 27
    gcd_t, 27
                                                         aerobus::Embed< zpz< x>, i32>, 52
    geometric sum, 27
                                                              type, 53
    Inp1, 27
                                                         aerobus::i32, 54
                                                              add t, 55
    make_frac_polynomial_t, 28
    make int polynomial t, 28
                                                              div t, 55
    make q32 t, 28
                                                              eq t, 55
    make_q64_t, 29
                                                              eq_v, 58
    makefraction_t, 29
                                                              gcd_t, 56
    mul t, 29
                                                              gt_t, 56
```

inject_constant_t, 56	aerobus::known_polynomials, 40
inject_ring_t, 56	hermite_kind, 40
inner_type, 56	physicist, 40
is_euclidean_domain, 58	probabilist, 40
is_field, 59	aerobus::polynomial< Ring >, 67
It_t, 57	add_t, 69
mod_t, 57	derive_t, 69
mul_t, 57	div_t, 69
one, 57	eq_t, 70
pos_t, <del>5</del> 7	gcd_t, 70
pos_v, 59	gt_t, 70
sub_t, 58	inject_constant_t, 71
zero, 58	inject_ring_t, 71
aerobus::i32::val $< x >$ , 86	is_euclidean_domain, 75
enclosing_type, 87	is_field, 75
get, 87	lt_t, 71
is_zero_t, 87	mod_t, 71
to_string, 87	monomial t, 72
v, 87	mul_t, 72
aerobus::i64, 59	one, 72
add_t, 61	pos_t, 72
div_t, 61	pos_v, 75
eq_t, 61	simplify_t, 74
eq_v, <del>64</del>	sub_t, 74
gcd_t, 61	X, 74
gt_t, 62	zero, 74
gt_v, 64	aerobus::polynomial< Ring >::horner_reduction_t< P
inject_constant_t, 62	>, 53
inject_ring_t, 62	aerobus::polynomial< Ring >::horner_reduction_t< P
inner_type, 62	>::inner< index, stop >, 65
is_euclidean_domain, 65	type, 66
is_field, 65	aerobus::polynomial< Ring >::horner_reduction_t< P
lt_t, 62	>::inner< stop, stop >, 66
It v, 65	type, 66
mod_t, 63	aerobus::polynomial< Ring >::val< coeffN >, 97
mul_t, 63	aN, 99
one, 63	coeff_at_t, 99
pos_t, 63	degree, 100
pos_v, 65	enclosing_type, 99
sub_t, 64	eval, 100
zero, 64	is_zero_t, 99
aerobus::i64::val $< x >$ , 88	is_zero_v, 100
enclosing_type, 89	ring_type, 99
get, 89	strip, 99
inner_type, 89	to_string, 100
is_zero_t, 89	value_at_t, 99
to_string, 89	aerobus::polynomial< Ring >::val< coeffN >::coeff_at<
v, 90	index, $E >$ , 43
aerobus::internal, 36	aerobus::polynomial< Ring >::val< coeffN >::coeff_at<
index_sequence_reverse, 40	index, std::enable_if_t<(index< 0     index >
is_instantiation_of_v, 40	0)>>, 43
make_index_sequence_reverse, 39	type, 43
type_at_t, 39	aerobus::polynomial < Ring >::val < coeffN >::coeff_at <
aerobus::is_prime< n >, 67	index, std::enable_if_t<(index==0)>>, 44
value, 67	type, 44
aerobus::IsEuclideanDomain, 41	aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN, coeffs $>$ ,
aerobus::IsField, 41	90
aerobus::IsRing, 42	aN, 91

coeff_at_t, 91	is_field, 107
degree, 93	lt_t, 104
enclosing_type, 91	lt_v, 107
eval, 92	mod_t, 104
is_zero_t, 92	mul_t, 105
is_zero_v, 93	one, 105
ring_type, 92	pos_t, 105
strip, 92	pos_v, 107
to_string, 93	sub_t, 105
value_at_t, 92	zero, 106
aerobus::Quotient< Ring, X >, 76	aerobus:: $zpz ::val < x >, 94$
add_t, 77	enclosing_type, 95
div_t, 78	get, 96
eq_t, 78	is_zero_t, 95
eq_v, 80	is_zero_v, 96
inject_constant_t, 78	to_string, 96
inject_ring_t, 78	v, 96
is_euclidean_domain, 80	aligned_malloc
mod_t, 79	aerobus, 34
mul_t, 79	alternate_t
one, 79	aerobus, 20
pos_t, 79	alternate_v
pos_v, 80	aerobus, 35
zero, 80	aN
aerobus::Quotient< Ring, X >::val< V >, 94	aerobus::polynomial $<$ Ring $>$ ::val $<$ coeffN $>$ , 99
raw_t, 94	aerobus::polynomial< Ring >::val< coeffN, coeffs
type, 94	>, 91
aerobus::type_list< Ts >, 82	asin
at, 83	aerobus, 21
concat, 83	asinh
insert, 83	aerobus, 21
length, 84	at
push_back, 83	aerobus::type_list< Ts >, 83
push_front, 84	atan
remove, 84	aerobus, 21
aerobus::type_list< Ts >::pop_front, 75	atanh
tail, 76	aerobus, 21
type, 76	bell t
aerobus::type_list< Ts >::split< index >, 81	<del>_</del>
head, 81	aerobus, 23 bernoulli t
tail, 81	aerobus, 23
aerobus::type_list<>>, 85	bernoulli_v
concat, 85	aerobus, 35
insert, 85	derobus, 55
length, 86	coeff_at_t
push_back, 85	aerobus::polynomial< Ring >::val< coeffN >, 99
push_front, 85	aerobus::polynomial< Ring >::val< coeffN, coeffs
aerobus::zpz, 101	>,91
add_t, 102	combination_t
div_t, 103	aerobus, 23
eq_t, 103	combination_v
eq_v, 106	aerobus, 35
gcd_t, 103	concat
gt_t, 103 gt_v, 106	aerobus::type_list< Ts >, 83
gt_v, 106 inject_constant_t, 104	aerobus::type_list<>, 85
inner_type, 104	cos
is_euclidean_domain, 107	aerobus, 23
is_cuonacan_aomain, 107	
	cosh

aerobus, 25	gcd_t aerobus, 27
degree	aerobus::i32, 56
aerobus::polynomial< Ring >::val< coeffN >, 100	aerobus::i64, 61
aerobus::polynomial< Ring >::val< coeffN, coeffs	aerobus::polynomial< Ring >, 70
>, 93	
derive_t	aerobus::zpz, 103
	geometric_sum
aerobus::polynomial < Ring >, 69	aerobus, 27
div_t	get
aerobus, 25	aerobus::i32::val $< x >$ , 87
aerobus::i32, 55	aerobus::i64::val $< x >$ , 89
aerobus::i64, 61	aerobus::zpz $<$ p $>$ ::val $<$ x $>$ , 96
aerobus::polynomial $<$ Ring $>$ , 69	gt_t
aerobus::Quotient $<$ Ring, $X >$ , 78	aerobus::i32, 56
aerobus::zpz $<$ p $>$ , 103	aerobus::i64, 62
	aerobus::polynomial< Ring >, 70
E_fraction	aerobus:: $zpz $ , 103
aerobus, 25	gt_v
embed_int_poly_in_fractions_t	aerobus::i64, 64
aerobus, 25	aerobus:: $zpz $ , 106
enclosing_type	αστουασ2β2 < β > , 100
aerobus::i32::val < $x >$ , 87	head
aerobus::i64::val < $x >$ , 89	aerobus::type_list< Ts >::split< index >, 81
aerobus::polynomial< Ring >::val< coeffN >, 99	hermite_kind
aerobus::polynomial< Ring >::val< coeffN, coeffs	aerobus::known_polynomials, 40
>, 91	aerobuskilowii_polyrioililais, 40
aerobus::zpz $<$ p $>$ ::val $<$ x $>$ , 95	index_sequence_reverse
eq_t	aerobus::internal, 40
aerobus::i32, 55	inject_constant_t
aerobus::i64, 61	aerobus::i32, 56
aerobus::polynomial < Ring >, 70	aerobus::i64, 62
aerobus::Quotient< Ring, X >, 78	aerobus::polynomial < Ring >, 71
aerobus::zpz, 103	aerobus::Quotient< Ring, X >, 78
eq_v	aerobus::zpz, 104
aerobus::i32, 58	inject_ring_t
aerobus::i64, 64	aerobus::i32, 56
aerobus::Quotient $<$ Ring, X $>$ , 80	aerobus::i64, 62
aerobus:: $zpz $ , 106	aerobus::polynomial $<$ Ring $>$ , 71
eval	aerobus::Quotient $<$ Ring, $X>$ , 78
aerobus::polynomial < Ring >::val < coeffN >, 100	inner_type
aerobus::polynomial< Ring >::val< coeffN, coeffs	aerobus::i32, 56
>, 92	aerobus::i64, 62
exp	aerobus:: $i64::val < x > , 89$
aerobus, 26	aerobus::zpz, 104
expm1	insert
aerobus, 26	aerobus::type_list< Ts >, 83
, -	aerobus::type list<>, 85
factorial_t	Introduction, 1
aerobus, 26	is_euclidean_domain
factorial_v	aerobus::i32, 58
aerobus, 35	aerobus::i64, 65
field	
aerobus, 34	aerobus::polynomial < Ring >, 75
	aerobus::Quotient< Ring, X >, 80
fpq32	aerobus::zpz, 107
aerobus, 26	is_field
fpq64	aerobus::i32, 59
aerobus, 27	aerobus::i64, 65
FractionField	aerobus::polynomial< Ring >, 75
aerobus, 27	aerobus::zpz $<$ p $>$ , 107

is_instantiation_of_v	one
aerobus::internal, 40	aerobus::i32, 57
is_zero_t	aerobus::i64, 63
aerobus::i32::val $<$ x $>$ , 87	aerobus::polynomial $<$ Ring $>$ , 72
aerobus::i64::val $<$ x $>$ , 89	aerobus::Quotient< Ring, X >, 79
aerobus::polynomial< Ring >::val< coeffN >, 99	aerobus::zpz $<$ p $>$ , 105
aerobus::polynomial< Ring >::val< coeffN, coeffs	mla valada
>, 92	physicist
aerobus::zpz ::val < x >, 95	aerobus::known_polynomials, 40 pi64
is_zero_v	aerobus, 30
aerobus::polynomial < Ring >::val < coeffN >, 100	PI_fraction
aerobus::polynomial< Ring >::val< coeffN, coeffs	aerobus, 30
>, 93	pos_t
aerobus::zpz::val< x >, 96	aerobus::i32, 57
length	aerobus::i64, 63
aerobus::type_list< Ts >, 84	aerobus::polynomial< Ring >, 72
aerobus::type_list<>>, 86	aerobus::Quotient< Ring, X >, 79
Inp1	aerobus::zpz $<$ p $>$ , 105
aerobus, 27	pos_v
lt_t	aerobus::i32, 59
aerobus::i32, 57	aerobus::i64, 65
aerobus::i64, 62	aerobus::polynomial < Ring >, 75
aerobus::polynomial < Ring >, 71	aerobus::Quotient< Ring, X >, 80
aerobus::zpz, 104	aerobus::zpz, 107
lt_v	pow_t
aerobus::i64, 65	aerobus, 30
aerobus::zpz, 107	pq64
	aerobus, 30
make_frac_polynomial_t	probabilist
aerobus, 28	aerobus::known_polynomials, 40
make_index_sequence_reverse	push_back
aerobus::internal, 39	aerobus::type_list< Ts >, 83
make_int_polynomial_t	aerobus::type_list<>, 85
aerobus, 28	push_front
make_q32_t	aerobus::type_list< Ts >, 84
aerobus, 28	aerobus::type_list<>, 85
make_q64_t	
aerobus, 29	q32
makefraction_t	aerobus, 30
aerobus, 29	q64
mod_t	aerobus, 31
aerobus::i32, 57	raw t
aerobus::i64, 63	aerobus::Quotient< Ring, X >::val< V >, 94
aerobus::polynomial < Ring >, 71	README.md, 109
aerobus::Quotient< Ring, X >, 79 aerobus::zpz, 104	remove
monomial t	aerobus::type_list< Ts >, 84
aerobus::polynomial < Ring >, 72	ring_type
mul t	aerobus::polynomial< Ring >::val< coeffN >, 99
aerobus, 29	aerobus::polynomial < Ring >::val < coeffN, coeffs
aerobus::i32, 57	>, 92
aerobus::i64, 63	,
aerobus::polynomial< Ring >, 72	simplify_t
aerobus::Quotient< Ring, X >, 79	aerobus::polynomial< Ring >, 74
aerobus:: $zpz $ , 105	sin
mulfractions_t	aerobus, 31
aerobus, 29	sinh
, -	aerobus, 31

```
SQRT2_fraction
                                                             aerobus::polynomial<
                                                                                     Ring
                                                                                             >::val<
                                                                                                       coeffN
     aerobus, 31
                                                                  >::coeff at<index, std::enable if t<(index==0)>
SQRT3 fraction
                                                                  >, 44
     aerobus, 31
                                                             aerobus::Quotient< Ring, X >::val< V >, 94
src/aerobus.h, 109
                                                             aerobus::type_list< Ts >::pop_front, 76
src/examples.h, 200
                                                        type at t
stirling 1 signed t
                                                             aerobus::internal, 39
     aerobus, 32
stirling 1 unsigned t
                                                             aerobus::i32::val < x >, 87
     aerobus, 32
                                                             aerobus::i64::val < x >, 90
stirling_2_t
                                                             aerobus::zpz ::val < x >, 96
     aerobus, 32
                                                        vadd t
strip
                                                             aerobus, 34
     aerobus::polynomial < Ring >::val < coeffN >, 99
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                        val
                                                             aerobus::ContinuedFraction < a0 >, 46
          >, 92
                                                             aerobus::ContinuedFraction < a0, rest... >, 47
sub_t
                                                        value
     aerobus, 33
                                                             aerobus::is prime < n >, 67
     aerobus::i32, 58
                                                        value at t
     aerobus::i64, 64
                                                             aerobus::polynomial < Ring >::val < coeffN >, 99
     aerobus::polynomial < Ring >, 74
                                                             aerobus::polynomial< Ring >::val< coeffN, coeffs
     aerobus::zpz , 105
                                                                  >, 92
tail
                                                        vmul t
     aerobus::type_list< Ts >::pop_front, 76
                                                             aerobus, 34
     aerobus::type_list< Ts >::split< index >, 81
                                                        Χ
tan
                                                             aerobus::polynomial < Ring >, 74
     aerobus, 33
tanh
                                                        zero
     aerobus, 33
                                                             aerobus::i32, 58
taylor
                                                             aerobus::i64, 64
     aerobus, 33
                                                             aerobus::polynomial < Ring >, 74
to_string
                                                             aerobus::Quotient < Ring, X >, 80
     aerobus::i32::val< x >, 87
                                                             aerobus::zpz, 106
     aerobus::i64::val< x >, 89
     aerobus::polynomial < Ring >::val < coeffN >, 100
     aerobus::polynomial< Ring >::val< coeffN, coeffs
         >, 93
     aerobus::zpz ::val < x >, 96
type
     aerobus::ContinuedFraction< a0 >, 45
     aerobus::ContinuedFraction < a0, rest... >, 47
     aerobus::Embed< i32, i64 >, 48
                          polynomial<
     aerobus::Embed<
                                          Small
                                                   >.
         polynomial < Large > >, 49
     aerobus::Embed< q32, q64 >, 50
     aerobus::Embed< Quotient< Ring, X >, Ring >,
     aerobus::Embed< Ring, FractionField< Ring >>,
     aerobus::Embed< zpz< x>, i32>, 53
     aerobus::polynomial < Ring >::horner reduction t <
         P > ::inner < index, stop >, 66
     aerobus::polynomial < Ring >::horner reduction t <
         P > ::inner < stop, stop >, 66
     aerobus::polynomial< Ring
                                    >::val< coeffN
          >::coeff_at< index, std::enable_if_t<(index<
         0 \mid | \text{index} > 0) > , 43
```