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

8.5 aerobus::ContinuedFraction $<$ a0 $>$ Struct Template Reference	47
8.5.1 Detailed Description	47
8.5.2 Member Typedef Documentation	47
8.5.2.1 type	47
8.5.3 Member Data Documentation	48
8.5.3.1 val	48
$8.6 \ aerobus:: Continued Fraction < a0, rest > Struct \ Template \ Reference \\ \ \ldots \\ \ \ldots \\ \ \ldots \\ \ \ldots$	48
8.6.1 Detailed Description	48
8.6.2 Member Typedef Documentation	49
8.6.2.1 type	49
8.6.3 Member Data Documentation	49
8.6.3.1 val	49
8.7 aerobus::ConwayPolynomial Struct Reference	49
8.8 aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner< index, ghost > Struct Template Reference	49
8.8.1 Member Function Documentation	50
8.8.1.1 func()	50
8.9 aerobus::polynomial < Ring >::compensated_horner < arithmeticType, P >::EFTHorner <-1, ghost > Struct Template Reference	50
8.9.1 Member Function Documentation	50
8.9.1.1 func()	50
8.10 aerobus::Embed< Small, Large, E > Struct Template Reference	51
8.10.1 Detailed Description	51
8.11 aerobus::Embed< i32, i64 > Struct Reference	51
8.11.1 Detailed Description	51
8.11.2 Member Typedef Documentation	51
8.11.2.1 type	51
8.12 aerobus::Embed< polynomial< Small $>$, polynomial< Large $>$ > Struct Template Reference	52
8.12.1 Detailed Description	52
8.12.2 Member Typedef Documentation	52
8.12.2.1 type	52
8.13 aerobus::Embed < q32, q64 > Struct Reference	53
8.13.1 Detailed Description	53
8.13.2 Member Typedef Documentation	53
8.13.2.1 type	53
8.14 aerobus::Embed< Quotient< Ring, X $>$, Ring $>$ Struct Template Reference	54
8.14.1 Detailed Description	54
8.14.2 Member Typedef Documentation	54
8.14.2.1 type	54
8.15 aerobus::Embed< Ring, FractionField< Ring $>>$ Struct Template Reference	55
8.15.1 Detailed Description	55
8.15.2 Member Typedef Documentation	55
8 15 2 1 type	55

8.19.2.11 mul_t	68
8.19.2.12 one	68
8.19.2.13 pos_t	68
8.19.2.14 sub_t	68
8.19.2.15 zero	69
8.19.3 Member Data Documentation	69
8.19.3.1 eq_v	69
8.19.3.2 gt_v	69
8.19.3.3 is_euclidean_domain	69
8.19.3.4 is_field	70
8.19.3.5 lt_v	70
8.19.3.6 pos_v	70
8.20 aerobus::polynomial < Ring >::horner_reduction_t < P >::inner < index, stop > Struct Template Reference	70
8.20.1 Member Typedef Documentation	71
8.20.1.1 type	71
$8.21 \ aerobus::polynomial < Ring > ::horner_reduction_t < P > ::inner < stop, \ stop > Struct \ Template \ Refusion = (a.2) \ Archive =$	
erence	71
8.21.1 Member Typedef Documentation	71
8.21.1.1 type	
8.22 aerobus::is_prime < n > Struct Template Reference	71
8.22.1 Detailed Description	72
8.22.2 Member Data Documentation	72
8.22.2.1 value	72
8.23 aerobus::polynomial < Ring > Struct Template Reference	72
8.23.1 Detailed Description	74
8.23.2 Member Typedef Documentation	74
8.23.2.1 add_t	74
8.23.2.2 derive_t	74
8.23.2.3 div_t	75
8.23.2.4 eq_t	75
8.23.2.5 gcd_t	75
8.23.2.6 gt_t	75
8.23.2.7 inject_constant_t	76
8.23.2.8 inject_ring_t	76
8.23.2.9 lt_t	76
8.23.2.10 mod_t	76
8.23.2.11 monomial_t	77
8.23.2.12 mul_t	77
8.23.2.13 one	77
8.23.2.14 pos_t	77
8.23.2.15 simplify_t	79
8.23.2.16 sub_t	79

8.23.2.17 X	79
8.23.2.18 zero	79
8.23.3 Member Data Documentation	80
8.23.3.1 is_euclidean_domain	80
8.23.3.2 is_field	80
8.23.3.3 pos_v	80
8.24 aerobus::type_list< Ts >::pop_front Struct Reference	80
8.24.1 Detailed Description	80
8.24.2 Member Typedef Documentation	81
8.24.2.1 tail	81
8.24.2.2 type	81
8.25 aerobus::Quotient $<$ Ring, X $>$ Struct Template Reference	81
8.25.1 Detailed Description	82
8.25.2 Member Typedef Documentation	82
8.25.2.1 add_t	82
8.25.2.2 div_t	83
8.25.2.3 eq_t	83
8.25.2.4 inject_constant_t	83
8.25.2.5 inject_ring_t	84
8.25.2.6 mod_t	84
8.25.2.7 mul_t	84
8.25.2.8 one	84
8.25.2.9 pos_t	85
8.25.2.10 zero	85
8.25.3 Member Data Documentation	85
8.25.3.1 eq_v	85
8.25.3.2 is_euclidean_domain	85
8.25.3.3 pos_v	85
8.26 aerobus::type_list< Ts >::split< index > Struct Template Reference	86
8.26.1 Detailed Description	86
8.26.2 Member Typedef Documentation	86
8.26.2.1 head	86
8.26.2.2 tail	86
8.27 aerobus::type_list< Ts > Struct Template Reference	87
8.27.1 Detailed Description	87
8.27.2 Member Typedef Documentation	88
8.27.2.1 at	88
8.27.2.2 concat	88
8.27.2.3 insert	88
8.27.2.4 push_back	89
8.27.2.5 push_front	89
8.27.2.6 remove	89

8.27.3 Member Data Documentation	89
8.27.3.1 length	89
8.28 aerobus::type_list<> Struct Reference	90
8.28.1 Detailed Description	90
8.28.2 Member Typedef Documentation	90
8.28.2.1 concat	90
8.28.2.2 insert	90
8.28.2.3 push_back	90
8.28.2.4 push_front	90
8.28.3 Member Data Documentation	91
8.28.3.1 length	91
8.29 aerobus::i32::val $<$ x $>$ Struct Template Reference	91
8.29.1 Detailed Description	91
8.29.2 Member Typedef Documentation	92
8.29.2.1 enclosing_type	92
8.29.2.2 is_zero_t	92
8.29.3 Member Function Documentation	92
8.29.3.1 get()	92
8.29.3.2 to_string()	92
8.29.4 Member Data Documentation	92
8.29.4.1 v	92
8.30 aerobus::i64::val < x > Struct Template Reference	93
8.30.1 Detailed Description	93
8.30.2 Member Typedef Documentation	94
8.30.2.1 enclosing_type	94
8.30.2.2 inner_type	94
8.30.2.3 is_zero_t	94
8.30.3 Member Function Documentation	94
8.30.3.1 get()	94
8.30.3.2 to_string()	94
8.30.4 Member Data Documentation	95
8.30.4.1 v	95
8.31 aerobus::polynomial< Ring >::val< coeffN, coeffs > Struct Template Reference	95
8.31.1 Detailed Description	96
8.31.2 Member Typedef Documentation	96
8.31.2.1 aN	96
8.31.2.2 coeff_at_t	96
8.31.2.3 enclosing_type	97
8.31.2.4 is_zero_t	97
8.31.2.5 ring_type	97
8.31.2.6 strip	97
8.31.2.7 value_at_t	97

8.31.3 Member Function Documentation	97
8.31.3.1 compensated_eval()	97
8.31.3.2 eval()	98
8.31.3.3 to_string()	98
8.31.4 Member Data Documentation	99
8.31.4.1 degree	99
8.31.4.2 is_zero_v	99
8.32 aerobus::Quotient $<$ Ring, $X>::$ val $<$ $V>$ Struct Template Reference	99
8.32.1 Detailed Description	99
8.32.2 Member Typedef Documentation	00
8.32.2.1 raw_t	00
8.32.2.2 type	00
8.33 aerobus::zpz::val< x > Struct Template Reference	00
8.33.1 Detailed Description	00
8.33.2 Member Typedef Documentation	01
8.33.2.1 enclosing_type	01
8.33.2.2 is_zero_t	01
8.33.3 Member Function Documentation	01
8.33.3.1 get()	01
8.33.3.2 to_string()	01
8.33.4 Member Data Documentation	02
8.33.4.1 is_zero_v	02
8.33.4.2 v	02
8.34 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference	02
8.34.1 Detailed Description	03
8.34.2 Member Typedef Documentation	03
8.34.2.1 aN	03
8.34.2.2 coeff_at_t	03
8.34.2.3 enclosing_type	03
8.34.2.4 is_zero_t	04
8.34.2.5 ring_type	04
8.34.2.6 strip	04
8.34.2.7 value_at_t	04
8.34.3 Member Function Documentation	04
8.34.3.1 compensated_eval()	04
8.34.3.2 eval()	04
8.34.3.3 to_string()	05
8.34.4 Member Data Documentation	05
8.34.4.1 degree	٥ ـ
0.04.4.0.1	υS
8.34.4.2 is_zero_v	
8.35 aerobus::zpz Struct Template Reference	05

8.35.2 Member Typedef Documentation	. 107
8.35.2.1 add_t	. 107
8.35.2.2 div_t	. 107
8.35.2.3 eq_t	. 108
8.35.2.4 gcd_t	. 108
8.35.2.5 gt_t	. 108
8.35.2.6 inject_constant_t	. 109
8.35.2.7 inner_type	. 109
8.35.2.8 lt_t	. 109
8.35.2.9 mod_t	. 109
8.35.2.10 mul_t	. 110
8.35.2.11 one	. 110
8.35.2.12 pos_t	. 110
8.35.2.13 sub_t	. 110
8.35.2.14 zero	. 111
8.35.3 Member Data Documentation	. 111
8.35.3.1 eq_v	. 111
8.35.3.2 gt_v	. 111
8.35.3.3 is_euclidean_domain	. 111
8.35.3.4 is_field	. 111
8.35.3.5 lt_v	. 112
8.35.3.6 pos_v	. 112
9 File Documentation	113
9.1 README.md File Reference	
9.1 README.md File Reference	
9.2 src/aerobus.h File Reierence	
9.4 src/examples.h File Reference	. 206
9.5 examples.n	. 206
10 Examples	207
10.1 examples/hermite.cpp	. 207
10.2 examples/custom_taylor.cpp	. 207
10.3 examples/fp16.cu	. 208
10.4 examples/continued_fractions.cpp	. 210
10.5 examples/modular_arithmetic.cpp	. 210
10.6 examples/make_polynomial.cpp	. 211
10.7 examples/polynomials_over_finite_field.cpp	. 211
10.8 examples/compensated_horner.cpp	. 212
Index	045
Index	215

Introduction

Aerobus is a C++-20 pure header library for general algebra on polynomials, discrete rings and associated structures.

Everything in Aerobus is expressed as types.

We say that again as it is the most fundamental characteristic of Aerobus:

Everything is expressed as types

The library serves two main purposes:

- Express algebra structures and associated operations in type arithmetic, compile-time;
- Provide portable and fast evaluation functions for polynomials.

It is designed to be 'quite easily' extensible.

Given these functions are "generated" at compile time and do not rely on inline assembly, they are actually platform independent, yielding exact same results if processors have same capabilities (such as Fused-Multiply-Add instructions).

1.1 HOW TO

- Clone or download the repository somewhere, or just download aerobus.h
- In your code, add: #include "aerobus.h"
- Compile with -std=c++20 (at least) -l<install_location>

Aerobus provides a definition for low-degree (up to 997) Conway polynomials. To use them, define AEROBUS — __CONWAY_IMPORTS before including aerobus.h.

2 Introduction

1.1.1 Unit Test

Install Cmake Install a recent compiler (supporting c++20), such as MSVC, G++ or Clang++

Move to the top directory then:

```
cmake -S . -B build
cmake --build build
cd build && ctest
```

Terminal should write:

100% tests passed, 0 tests failed out of 48

Alternate way:

make tests

From top directory.

1.1.2 Benchmarks

Benchmarks are written for Intel CPUs having AVX512f and AVX512vl flags, they work only on Linux operating system using g++.

In addition of Cmake and compiler, install OpenMP. And Google's Benchmark library. Then move to top directory:

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

1.2 Structures

1.2.1 Predefined discrete euclidean domains

Aerobus predefines several simple euclidean domains, such as :

```
aerobus::i32:integers (32 bits)
aerobus::i64:integers (64 bits)
aerobus::zpz:integers modulo p (prime number) on 32 bits
```

All these types represent the Ring, meaning the algebraic structure. They have a nested type val < i > where i is a scalar native value (int32_t or int64_t) to represent actual values in the ring. They have the following "operations", required by the IsEuclideanDomain concept :

- add_t : a type (specialization of val), representing addition between two values
- $\bullet \ \, \mathrm{sub_t}: a \ type \ (specialization \ of \ val), \ representing \ subtraction \ between \ two \ values$
- mul_t : a type (specialization of val), representing multiplication between two values
- div_t : a type (specialization of val), representing division between two values
- mod_t : a type (specialization of val), representing modulus between two values

and the following "elements":

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

1.2 Structures 3

1.2.2 Polynomials

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

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

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

1.2.3 Known polynomials

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

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

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

1.2.4 Conway polynomials

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

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

4 Introduction

1.2.5 Taylor series

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

They can be used and evaluated:

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

Exposed functions are:

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

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

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

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

1.3 Operations 5

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

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

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

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

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

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

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

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

1.3 Operations

1.3.1 Field of fractions

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

6 Introduction

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

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

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

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

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

Which also have an evaluation function, as polynomial do.

1.3.2 Quotient

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

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

As we could have using zpz<17>.

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

1.4 Misc

1.4.1 Continued Fractions

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

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

1.5 CUDA

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

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

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

Please push to make these bug fixed by NVIDIA.

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

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

8 Namespace Index

Concept Index

3.1 Concepts

Here is a list of all concepts with brief descriptions:

aerobus::IsEuclideanDomain	
Concept to express R is an euclidean domain	43
aerobus::IsField	
Concept to express R is a field	43
aerobus::IsRing	
Concept to express B is a Bing	44

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 >	45
aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable_if_t < (index < 0 index > 0) > 3	>
aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)>>	46
aerobus::ContinuedFraction < values >	
Continued fraction a0 + $\frac{1}{a_1 + \frac{1}{a_2 + \dots}}$	46
aerobus::ContinuedFraction $<$ a0 $>$	
Specialization for only one coefficient, technically just 'a0'	47
aerobus::ContinuedFraction< a0, rest >	
Specialization for multiple coefficients (strictly more than one)	48
aerobus::ConwayPolynomial	49
aerobus::polynomial < Ring >::compensated_horner < arithmeticType, P >::EFTHorner < index, ghost >	49
aerobus::polynomial < Ring >::compensated_horner < arithmeticType, P >::EFTHorner <-1, ghost >	50
aerobus::Embed< Small, Large, E >	
Embedding - struct forward declaration	51
aerobus::Embed< i32, i64 >	
Embeds i32 into i64	51
aerobus::Embed< polynomial< Small >, polynomial< Large > >	
Embeds polynomial <small> into polynomial<large></large></small>	52
aerobus::Embed < q32, q64 >	
Embeds q32 into q64	53
aerobus::Embed< Quotient< Ring, X >, Ring >	
Embeds Quotient < Ring, X > into Ring	54
aerobus::Embed < Ring, FractionField < Ring > >	
Embeds values from Ring to its field of fractions	55
aerobus::Embed < zpz < x >, i32 >	
Embeds zpz values into i32	55
aerobus::polynomial < Ring >::horner_reduction_t < P >	
Used to evaluate polynomials over a value in Ring	56
aerobus::i32	
32 bits signed integers, seen as a algebraic ring with related operations	57
aerobus::i64	
64 bits signed integers, seen as a algebraic ring with related operations	64
aerobus::polynomial< Ring >::horner_reduction_t< P >::inner< index, stop >	70
aerobus::polynomial < Ring >::horner_reduction_t < P >::inner < stop, stop >	71

12 Class Index

aerobus::is_prime< n >	
Checks if n is prime	71
aerobus::polynomial < Ring >	72
aerobus::type_list< Ts >::pop_front	
Removes types from head of the list	80
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	81
aerobus::type_list< Ts >::split< index >	
Splits list at index	86
aerobus::type_list< Ts >	
Empty pure template struct to handle type list	87
aerobus::type_list<>	
Specialization for empty type list	90
aerobus::i32::val< x >	
Values in i32, again represented as types	91
aerobus::i64::val< x >	
Values in i64	93
aerobus::polynomial < Ring >::val < coeffN, coeffs >	
Values (seen as types) in polynomial ring	95
aerobus::Quotient < Ring, X >::val < V >	
Projection values in the quotient ring	99
aerobus::zpz::val< x >	
Values in zpz	100
aerobus::polynomial < Ring >::val < coeffN >	
Specialization for constants	102
aerobus::zpz	
Congruence classes of integers modulo p (32 bits)	105

File Index

5.1 File List

Here is a list of all files with brief descriptions:

src/aerobus.h .							 																	113
src/examples.h							 																	206

14 File Index

Namespace Documentation

6.1 aerobus Namespace Reference

main namespace for all publicly exposed types or functions

Namespaces

- · namespace internal
 - internal implementations, subject to breaking changes without notice
- namespace known_polynomials

families of well known polynomials such as Hermite or Bernstein

Classes

```
• struct ContinuedFraction
```

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

struct ContinuedFraction < a0 >

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

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

```
embedding - struct forward declaration
```

struct Embed< i32, i64 >

embeds i32 into i64

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

embeds polynomial<Small> into polynomial<Large>

struct Embed< q32, q64 >

embeds q32 into q64

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

embeds Quotient<Ring, X> into Ring

struct Embed< Ring, FractionField< Ring > >

embeds values from Ring to its field of fractions

struct Embed< zpz< x >, i32 >

embeds zpz values into i32

• struct i32

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

struct i64

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

• struct is_prime

checks if n is prime

- struct polynomial
- struct Quotient

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

struct type list

Empty pure template struct to handle type list.

struct type_list<>

specialization for empty type list

struct zpz

congruence classes of integers modulo p (32 bits)

Concepts

· concept IsRing

Concept to express R is a Ring.

• concept IsEuclideanDomain

Concept to express R is an euclidean domain.

concept IsField

Concept to express R is a field.

Typedefs

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

    template<typename val >

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

    template<typename Ring >

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

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

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

    template<typename X , typename Y >

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

 using q32 = FractionField < i32 >

     32 bits rationals rationals with 32 bits numerator and denominator

    using fpq32 = FractionField< polynomial< q32 >>

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

 using q64 = FractionField < i64 >

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

    using fpq64 = FractionField< polynomial< q64 > >

     polynomial with 64 bits rational coefficients

    template<typename Ring , typename v1 , typename v2 >

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

    template<typename Ring , typename v1 , typename v2 >

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

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

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

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

    template<typename Integers , size_t deg>

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

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

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

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

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

Functions

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

Variables

6.1.1 Detailed Description

main namespace for all publicly exposed types or functions

6.1.2 Typedef Documentation

6.1.2.1 abs t

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

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

Template Parameters

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

6.1.2.2 add_t

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

generic addition

Template Parameters

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

6.1.2.3 addfractions_t

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

helper type: adds two fractions

Template Parameters

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

6.1.2.4 alternate_t

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

Template Parameters

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

6.1.2.5 asin

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

Template Parameters

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

6.1.2.6 asinh

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

Template Parameters

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

6.1.2.7 atan

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

Template Parameters

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

6.1.2.8 atanh

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

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

Template Parameters

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

6.1.2.9 bell_t

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

Bell numbers.

Template Parameters

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

6.1.2.10 bernoulli_t

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

nth bernoulli number as type in T

Template Parameters

T	Ring type (i64)
n	

6.1.2.11 combination_t

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

computes binomial coefficient (k among n) as type

Template Parameters

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

6.1.2.12 cos

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

 $\cos(x)$ cosinus

Template Parameters

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

6.1.2.13 cosh

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

Template Parameters

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

6.1.2.14 div_t

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

generic division

Template Parameters

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

6.1.2.15 E_fraction

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

approximation of \boldsymbol{e}

6.1.2.16 embed_int_poly_in_fractions_t

embed a polynomial with integers coefficients into rational coefficients polynomials

Lives in polynomial<FractionField<Ring>>

Template Parameters

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

6.1.2.17 exp

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

Template Parameters

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

6.1.2.18 expm1

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

Template Parameters

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

6.1.2.19 factorial_t

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

computes factorial(i), as type

Template Parameters

Т	Ring type (e.g. i32)
i	

6.1.2.20 fpq32

```
using aerobus::fpq32 = typedef FractionField<polynomial<q32> >
```

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

6.1.2.21 fpq64

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

polynomial with 64 bits rational coefficients

6.1.2.22 FractionField

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

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

Template Parameters

```
Ring
```

6.1.2.23 gcd t

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

computes the greatest common divisor or A and B

Template Parameters

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

6.1.2.24 geometric_sum

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

Template Parameters

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

6.1.2.25 Inp1

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

Template Parameters

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

6.1.2.26 make_frac_polynomial_t

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

make a polynomial with coefficients in FractionField<Ring>

Template Parameters

Ring	integers
xs	values

6.1.2.27 make_int_polynomial_t

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

make a polynomial with coefficients in Ring

Template Parameters

Ring	integers
xs	coefficients

6.1.2.28 make_q32_t

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

helper type: make a fraction from numerator and denominator

Template Parameters

р	numerator
q	denominator

6.1.2.29 make_q64_t

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

helper type: make a fraction from numerator and denominator

Template Parameters

р	numerator
q	denominator

6.1.2.30 makefraction_t

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

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

Template Parameters

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

6.1.2.31 mul_t

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

generic multiplication

Template Parameters

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

6.1.2.32 mulfractions_t

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

helper type: multiplies two fractions

Template Parameters

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

6.1.2.33 pi64

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

polynomial with 64 bits integers coefficients

6.1.2.34 PI_fraction

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

representation of π as a continued fraction

6.1.2.35 pow_t

```
template<typename T , typename p , size_t n>
using aerobus::pow_t = typedef typename internal::pow<T, p, n>::type
```

 p^n (as 'val' type in T)

Template Parameters

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

6.1.2.36 pq64

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

polynomial with 64 bits rationals coefficients

6.1.2.37 q32

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

32 bits rationals rationals with 32 bits numerator and denominator

6.1.2.38 q64

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

64 bits rationals rationals with 64 bits numerator and denominator

6.1.2.39 sin

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

Template Parameters

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

6.1.2.40 sinh

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

Template Parameters

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

6.1.2.41 SQRT2_fraction

approximation of $\sqrt{2}$

6.1.2.42 SQRT3_fraction

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

approximation of

6.1.2.43 stirling_1_signed_t

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

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

Template Parameters

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

6.1.2.44 stirling_1_unsigned_t

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

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

Template Parameters

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

6.1.2.45 stirling_2_t

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

Stirling number of second king – as types.

Template Parameters

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

6.1.2.46 sub_t

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

generic subtraction

Template Parameters

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

6.1.2.47 tan

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

Template Parameters

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

6.1.2.48 tanh

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

tanh(x) hyperbolic tangent

Template Parameters

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

6.1.2.49 taylor

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

Template Parameters

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

Generated by Doxygen

6.1.2.50 vadd_t

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

adds multiple values (v1 + v2 + \dots + vn) vals must have same "enclosing_type" and "enclosing_type" must have an add_t binary operator

Template Parameters

```
...vals
```

6.1.2.51 vmul_t

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

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

Template Parameters



6.1.3 Function Documentation

6.1.3.1 aligned_malloc()

'portable' aligned allocation of count elements of type T

Template Parameters

T the type of elements to store

Parameters

count	the number of elements
alignment	boundary

6.1.3.2 field()

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

prime number)

6.1.4 Variable Documentation

6.1.4.1 alternate v

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

(-1)[∧]k as value from T

Template Parameters

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

6.1.4.2 bernoulli_v

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

nth bernoulli number as value in FloatType

Template Parameters

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

6.1.4.3 combination_v

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

computes binomial coefficients (k among n) as value

Template Parameters

T	(aerobus::i32 for example)
k	
n	

6.1.4.4 factorial_v

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

computes factorial(i) as value in T

Template Parameters

T	(aerobus::i64 for example)
i	

6.2 aerobus::internal Namespace Reference

internal implementations, subject to breaking changes without notice

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

Classes

```
    struct FractionField

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

· struct _is_prime
struct _is_prime< 0, i >

    struct _is_prime< 1, i >

• struct _{\mbox{is\_prime}}< 2, i >

    struct _is_prime< 3, i >

    struct _is_prime< 5, i >

• struct _{\bf is\_prime}< 7, i >

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

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

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

    struct AbelHelper

• struct AllOneHelper

    struct AllOneHelper< 0, I >

· struct alternate

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

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

· struct asin_coeff

    struct asin_coeff_helper

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

 struct atan_coeff struct atan_coeff_helper struct atan_coeff_helper< T, i, std::enable_if_t<(i &1)==0 >> struct atan_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >> · struct atanh coeff struct atanh_coeff_helper struct atanh coeff helper< T, i, std::enable if t<(i &1)==0>> struct atanh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >> struct bell_helper • struct bell_helper< T, 0 > struct bell_helper< T, 1 > struct bell_helper< T, n, std::enable_if_t<(n > 1)>> struct bernoulli struct bernoulli < T, 0 > • struct bernoulli_coeff • struct bernoulli helper struct bernoulli_helper< T, accum, m, m > struct bernstein helper struct bernstein_helper< 0, 0, I > • struct bernstein_helper< i, m, l, std::enable_if_t<(m > 0) &&(i > 0) &&(i < m)> > struct bernstein_helper< i, m, I, std::enable_if_t<(m > 0) &&(i==0)> > struct bernstein_helper< i, m, I, std::enable_if_t<(m > 0) &&(i==m)> > • struct BesselHelper struct BesselHelper< 0, I > struct BesselHelper< 1, I > struct chebyshev_helper struct chebyshev_helper< 1, 0, I > struct chebyshev_helper< 1, 1, I > struct chebyshev helper< 2, 0, I > struct chebyshev_helper< 2, 1, I > • struct combination struct combination helper struct combination_helper< T, 0, n > • struct combination_helper< T, k, n, std::enable_if_t<(n>=0 &&k >(n/2) &&k > 0)> > struct combination_helper< T, k, n, std::enable_if_t<(n >=0 &&k<=(n/2) &&k > 0)> > struct cos_coeff struct cos coeff helper struct cos_coeff_helper< T, i, std::enable_if_t<(i &1)==0 >> struct cos_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >> struct cosh_coeff · struct cosh coeff helper struct cosh_coeff_helper< T, i, std::enable_if_t<(i &1)==0 >> struct cosh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >> struct exp_coeff · struct factorial struct factorial < T, 0 > struct factorial< T, x, std::enable_if_t<(x > 0)>> struct FloatLayout struct FloatLayout< double > struct FloatLayout< float > struct FloatLayout< long double > struct fma helper struct fma_helper< double > struct fma_helper< float >

struct fma_helper< int16_t >

```
    struct fma_helper< int32_t >

    struct fma_helper< int64_t >

    struct fma_helper< long double >

    struct FractionFieldImpl

    struct FractionFieldImpl< Field, std::enable if t< Field::is field >>

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

· struct gcd
     greatest common divisor computes the greatest common divisor exposes it in gcd<A, B>::type as long as Ring type
     is an integral domain
- struct gcd< Ring, std::enable_if_t< Ring::is_euclidean_domain > >

    struct geom coeff

• struct hermite_helper

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

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

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

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

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

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

• struct insert h
· struct is instantiation of

    struct is_instantiation_of< TT, TT< Ts... >>

    struct laguerre helper

    struct 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

represented value as aerobus::q64

8.5.2.1 type

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

8.5.3 Member Data Documentation

8.5.3.1 val

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

represented value as double

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

· src/aerobus.h

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

specialization for multiple coefficients (strictly more than one)

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

Public Types

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

Static Public Attributes

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

8.6.1 Detailed Description

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

specialization for multiple coefficients (strictly more than one)

Template Parameters

a0	integer (int64_t)
rest	integers (int64_t)

8.6.2 Member Typedef Documentation

8.6.2.1 type

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

represented value as aerobus::q64

8.6.3 Member Data Documentation

8.6.3.1 val

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

reprensented value as double

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

• src/aerobus.h

8.7 aerobus::ConwayPolynomial Struct Reference

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

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

src/aerobus.h

8.8 aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P >::EFTHorner< index, ghost > Struct Template Reference

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

Static Public Member Functions

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

8.8.1 Member Function Documentation

8.8.1.1 func()

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

· src/aerobus.h

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

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

Static Public Member Functions

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

8.9.1 Member Function Documentation

8.9.1.1 func()

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

• src/aerobus.h

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

embedding - struct forward declaration

8.10.1 Detailed Description

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

embedding - struct forward declaration

Template Parameters

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

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

• src/aerobus.h

8.11 aerobus::Embed< i32, i64 > Struct Reference

```
embeds i32 into i64
```

#include <aerobus.h>

Public Types

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

8.11.1 Detailed Description

embeds i32 into i64

8.11.2 Member Typedef Documentation

8.11.2.1 type

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

the i64 representation of val

Template Parameters

```
val a value in i32
```

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

• src/aerobus.h

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

```
embeds polynomial<Small> into polynomial<Large>
```

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

Public Types

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

the polynomial<Large> reprensentation of v

8.12.1 Detailed Description

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

embeds polynomial<Small> into polynomial<Large>

Template Parameters

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

8.12.2 Member Typedef Documentation

8.12.2.1 type

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

the polynomial<Large> reprensentation of v

Template Parameters

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

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

· src/aerobus.h

8.13 aerobus::Embed < q32, q64 > Struct Reference

```
embeds q32 into q64
```

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

Public Types

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

8.13.1 Detailed Description

embeds q32 into q64

8.13.2 Member Typedef Documentation

8.13.2.1 type

```
template<typename v > using aerobus::Embed< q32, q64 >::type = make_q64_t<static_cast<int64_t>(v::x::v), static_\leftarrow cast<int64_t>(v::y::v)>
```

q64 representation of v

Template Parameters

```
v a value in q32
```

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

• src/aerobus.h

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

embeds Quotient<Ring, X> into Ring

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

Public Types

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

8.14.1 Detailed Description

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

embeds Quotient<Ring, X> into Ring

Template Parameters

Ring	a Euclidean ring
X	a value in Ring

8.14.2 Member Typedef Documentation

8.14.2.1 type

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

Ring reprensentation of val.

Template Parameters

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

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

• src/aerobus.h

8.15 aerobus::Embed< Ring, FractionField< Ring > > Struct Template Reference

embeds values from Ring to its field of fractions

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

Public Types

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

8.15.1 Detailed Description

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

embeds values from Ring to its field of fractions

Template Parameters

Ring an integers ring, such as i32

8.15.2 Member Typedef Documentation

8.15.2.1 type

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

FractionField<Ring> reprensentation of v.

Template Parameters

```
v a Ring value
```

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

• src/aerobus.h

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

embeds zpz values into i32

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

Public Types

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

8.16.1 Detailed Description

8.16.2 Member Typedef Documentation

8.16.2.1 type

an integer

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

the i32 reprensentation of val

Template Parameters

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

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

• src/aerobus.h

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

Used to evaluate polynomials over a value in Ring.

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

Classes

- struct inner
- struct inner< stop, stop >

8.17.1 Detailed Description

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

Used to evaluate polynomials over a value in Ring.

Template Parameters

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

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

· src/aerobus.h

8.18 aerobus::i32 Struct Reference

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

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

Classes

• struct val values in i32, again represented as types

Public Types

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

    template<typename v1 , typename v2 >

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

    template<typename v1 , typename v2 >

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

    template<typename v1 , typename v2 >

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

    template<typename v1 , typename v2 >

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

    template<typename v1 , typename v2 >

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

Static Public Attributes

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

8.18.1 Detailed Description

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

Examples

examples/compensated_horner.cpp.

8.18.2 Member Typedef Documentation

8.18.2.1 add t

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

addition operator yields v1 + v2

Template Parameters

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

8.18.2.2 div_t

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

division operator yields v1 / v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.3 eq_t

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

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

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.4 gcd_t

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

greatest common divisor yields GCD(v1, v2)

Template Parameters

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

8.18.2.5 gt t

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

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

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.6 inject_constant_t

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

inject a native constant

Template Parameters



8.18.2.7 inject_ring_t

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

8.18.2.8 inner_type

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

8.18.2.9 It t

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

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

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.10 mod_t

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

modulus operator yields v1 % v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.11 mul_t

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

multiplication operator yields v1 * v2

Template Parameters

v1	/1 a value in i32	
v2	a value in i32	

8.18.2.12 one

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

constant one

8.18.2.13 pos_t

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

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

Template Parameters

```
v a value in i32
```

8.18.2.14 sub_t

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

substraction operator yields v1 - v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.18.2.15 zero

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

constant zero

8.18.3 Member Data Documentation

8.18.3.1 eq_v

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

equality operator (boolean value)

Template Parameters

v1	
v2	

8.18.3.2 is_euclidean_domain

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

integers are an euclidean domain

8.18.3.3 is_field

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

integers are not a field

8.18.3.4 pos_v

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

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

Template Parameters

```
v a value in i32
```

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

· src/aerobus.h

8.19 aerobus::i64 Struct Reference

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

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

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

Classes

struct val

values in i64

Public Types

```
• using inner type = int64 t
     type of represented values
template<auto x>
  using inject_constant_t = val< static_cast< int64_t >(x)>
     injects constant as an i64 value

    template<typename v >

  using inject_ring_t = v
     injects a value used for internal consistency and quotient rings implementations for example i64::inject_ring_t<i64::val<1>>
      -> i64::val<1>
• using zero = val< 0 >
     constant zero
• using one = val< 1 >
     constant one

    template<typename v1 , typename v2 >

  using add t = typename add< v1, v2 >::type
     addition operator

    template<typename v1 , typename v2 >

  using sub_t = typename sub< v1, v2 >::type
     substraction operator

    template<typename v1 , typename v2 >

  using mul_t = typename mul < v1, v2 >::type
     multiplication operator
• template<typename v1 , typename v2 >
  using div_t = typename div < v1, v2 >::type
     division operator integer division
• template<typename v1 , typename v2 >
```

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

    template<typename v >

  using pos_t = typename pos< v >::type
     is v posititive yields v > 0 as std::true_type or std::false_type
```

Static Public Attributes

```
    static constexpr bool is_field = false
        integers are not a field
    static constexpr bool is_euclidean_domain = true
        integers are an euclidean domain
    template<typename v1, typename v2 >
        static constexpr bool gt_v = gt_t<v1, v2>::value
            strictly greater operator yields v1 > v2 as boolean value
    template<typename v1, typename v2 >
        static constexpr bool lt_v = lt_t<v1, v2>::value
            strictly smaller operator yields v1 < v2 as boolean value</li>
    template<typename v1, typename v2 >
        static constexpr bool eq_v = eq_t<v1, v2>::value
            equality operator yields v1 == v2 as boolean value
    template<typename v >
            static constexpr bool pos_v = pos_t<v>::value
```

8.19.1 Detailed Description

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

8.19.2 Member Typedef Documentation

positivity yields v > 0 as boolean value

8.19.2.1 add t

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

Template Parameters

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

8.19.2.2 div_t

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

division operator integer division

Template Parameters

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

8.19.2.3 eq_t

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

equality operator yields v1 == v2 as std::true_type or std::false_type

Template Parameters

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

8.19.2.4 gcd_t

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

greatest common divisor yields GCD(v1, v2) as instanciation of i64::val

Template Parameters

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

8.19.2.5 gt_t

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

strictly greater operator yields v1 > v2 as std::true_type or std::false_type

Template Parameters

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

8.19.2.6 inject_constant_t

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

injects constant as an i64 value

Template Parameters



8.19.2.7 inject_ring_t

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

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

Template Parameters

```
v a value in i64
```

8.19.2.8 inner_type

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

type of represented values

8.19.2.9 lt_t

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

strict less operator yields v1 < v2 as std::true_type or std::false_type

Template Parameters

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

8.19.2.10 mod_t

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

modulus operator

Template Parameters

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

8.19.2.11 mul_t

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

multiplication operator

Template Parameters

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

8.19.2.12 one

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

constant one

8.19.2.13 pos_t

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

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

Template Parameters

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

8.19.2.14 sub_t

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

substraction operator

Template Parameters

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

8.19.2.15 zero

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

constant zero

8.19.3 Member Data Documentation

8.19.3.1 eq_v

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

equality operator yields v1 == v2 as boolean value

Template Parameters

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

8.19.3.2 gt_v

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

strictly greater operator yields v1 > v2 as boolean value

Template Parameters

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

8.19.3.3 is_euclidean_domain

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

integers are an euclidean domain

8.19.3.4 is_field

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

integers are not a field

8.19.3.5 It v

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

strictly smaller operator yields v1 < v2 as boolean value

Template Parameters

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

8.19.3.6 pos_v

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

positivity yields v>0 as boolean value

Template Parameters

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

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

• src/aerobus.h

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

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

Public Types

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

8.20.1 Member Typedef Documentation

8.20.1.1 type

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

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

• src/aerobus.h

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

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

Public Types

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

8.21.1 Member Typedef Documentation

8.21.1.1 type

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

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

• src/aerobus.h

8.22 aerobus::is_prime < n > Struct Template Reference

checks if n is prime

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

Static Public Attributes

static constexpr bool value = internal::_is_prime<n, 5>::value
 true iff n is prime

8.22.1 Detailed Description

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

Template Parameters
```

8.22.2 Member Data Documentation

8.22.2.1 value

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

true iff n is prime

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

• src/aerobus.h

8.23 aerobus::polynomial < Ring > Struct Template Reference

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

Classes

• struct horner_reduction_t

Used to evaluate polynomials over a value in Ring.

struct val

values (seen as types) in polynomial ring

struct val< coeffN >

specialization for constants

Public Types

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

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

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

     generator

    template<typename P >

  using simplify t = typename simplify < P >::type
     simplifies a polynomial (recursively deletes highest degree if zero, do nothing otherwise)

    template<typename v1 , typename v2 >

  using add_t = typename add< v1, v2 >::type
     adds two polynomials

    template<typename v1 , typename v2 >

  using sub_t = typename sub< v1, v2 >::type
     substraction of two polynomials
• template<typename v1 , typename v2 >
  using mul_t = typename mul < v1, v2 >::type
     multiplication of two polynomials
• template<typename v1 , typename v2 >
  using eq_t = typename eq_helper< v1, v2 >::type
     equality operator
• template<typename v1 , typename v2 >
  using It_t = typename It_helper< v1, v2 >::type
     strict less operator
• template<typename v1, typename v2 >
  using gt_t = typename gt_helper< v1, v2 >::type
     strict greater operator
• template<typename v1 , typename v2 >
  using div_t = typename div < v1, v2 >::q_type
     division operator
• template<typename v1 , typename v2 >
  using mod_t = typename div_helper< v1, v2, zero, v1 >::mod_type
     modulo operator
• template<typename coeff , size_t deg>
  using monomial_t = typename monomial < coeff, deg >::type
     monomial: coeff X^{\wedge} deg

    template<typename v >

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

    template<typename v >

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

    template<typename v1 , typename v2 >

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

    template<auto x>

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

    template<typename v >

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

Static Public Attributes

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

```
• static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain
```

```
    template < typename v >
        static constexpr bool pos_v = pos_t < v > ::value
        positivity operator
```

8.23.1 Detailed Description

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

polynomial with coefficients in Ring Ring must be an integral domain

Examples

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

8.23.2 Member Typedef Documentation

8.23.2.1 add_t

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

adds two polynomials

Template Parameters

v1	
v2	

8.23.2.2 derive_t

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

derivation operator

Template Parameters



8.23.2.3 div_t

```
template<typename Ring >
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring >::div_t = typename div<v1, v2>::q_type
```

division operator

Template Parameters

v1	
v2	

8.23.2.4 eq_t

```
template<typename Ring >
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring >::eq_t = typename eq_helper<v1, v2>::type
```

equality operator

Template Parameters

v1	
v2	

8.23.2.5 gcd_t

```
template<typename Ring >
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring >::gcd_t = std::conditional_t< Ring::is_euclidean_domain,
typename make_unit<gcd_t<polynomial<Ring>, v1, v2> >::type, void>
```

greatest common divisor of two polynomials

Template Parameters

v1	
v2	

8.23.2.6 gt_t

```
template<typename Ring >
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring >::gt_t = typename gt_helper<v1, v2>::type
```

strict greater operator

Template Parameters

v1	
v2	

8.23.2.7 inject constant t

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

makes the constant (native type) polynomial a_0

Template Parameters



8.23.2.8 inject_ring_t

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

makes the constant (ring type) polynomial a_0

Template Parameters



8.23.2.9 lt_t

```
template<typename Ring >
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring >::lt_t = typename lt_helper<v1, v2>::type
```

strict less operator

Template Parameters

v1	
V 1	
v2	

8.23.2.10 mod t

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

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

modulo operator

Template Parameters

v1	
v2	

8.23.2.11 monomial_t

```
template<typename Ring >
template<typename coeff , size_t deg>
using aerobus::polynomial< Ring >::monomial_t = typename monomial<coeff, deg>::type
```

monomial : coeff X^deg

Template Parameters

coeff	
deg	

8.23.2.12 mul_t

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

multiplication of two polynomials

Template Parameters

v1	
v2	

8.23.2.13 one

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

constant one

8.23.2.14 pos_t

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

checks for positivity (an > 0)

٦	Гет	n	la	ŀρ	Pa	ra	m	ρĺ	Ъ	rs

V	
---	--

8.23.2.15 simplify_t

```
template<typename Ring >
template<typename P >
using aerobus::polynomial< Ring >::simplify_t = typename simplify<P>::type
```

simplifies a polynomial (recursively deletes highest degree if zero, do nothing otherwise)

Template Parameters



8.23.2.16 sub_t

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

substraction of two polynomials

Template Parameters

v1	
v2	

8.23.2.17 X

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

generator

8.23.2.18 zero

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

constant zero

8.23.3 Member Data Documentation

8.23.3.1 is euclidean domain

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

8.23.3.2 is field

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

8.23.3.3 pos_v

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

positivity operator

Template Parameters

```
v a value in polynomial::val
```

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

· src/aerobus.h

8.24 aerobus::type_list< Ts >::pop_front Struct Reference

removes types from head of the list

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

Public Types

- using type = typename internal::pop_front_h< Ts... >::head
 type that was previously head of the list
- using tail = typename internal::pop_front_h< Ts... >::tail remaining types in parent list when front is removed

8.24.1 Detailed Description

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

removes types from head of the list

8.24.2 Member Typedef Documentation

8.24.2.1 tail

```
template<typename... Ts>
using aerobus::type_list< Ts >::pop_front::tail = typename internal::pop_front_h<Ts...>::tail
```

remaining types in parent list when front is removed

8.24.2.2 type

```
template<typename... Ts>
using aerobus::type_list< Ts >::pop_front::type = typename internal::pop_front_h<Ts...>::head
```

type that was previously head of the list

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

· src/aerobus.h

8.25 aerobus::Quotient < Ring, X > Struct Template Reference

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

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

Classes

 struct val projection values in the quotient ring

Public Types

```
    using zero = val< typename Ring::zero > zero value
    using one = val< typename Ring::one > one
    template<typename v1 , typename v2 > using add_t = val< typename Ring::template add_t< typename v1::type, typename v2::type > > addition operator
    template<typename v1 , typename v2 > using mul_t = val< typename Ring::template mul_t< typename v1::type, typename v2::type > > substraction operator
    template<typename v1 , typename v2 > using div_t = val< typename Ring::template div_t< typename v1::type, typename v2::type > > division operator
    template<typename v1 , typename Ring::template div_t< typename v1::type, typename v2::type > > division operator
    template<typename v1 , typename v2 >
    template<typename v1 , typename v2 >
```

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

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

Static Public Attributes

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

8.25.1 Detailed Description

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

quotien rings are euclidean domain

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

Template Parameters

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

8.25.2 Member Typedef Documentation

8.25.2.1 add_t

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

addition operator

Template Parameters

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

8.25.2.2 div t

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

division operator

Template Parameters

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

8.25.2.3 eq_t

```
template<typename Ring , typename X >
template<typename v1 , typename v2 >
using aerobus::Quotient< Ring, X >::eq_t = typename Ring::template eq_t<typename v1::type,
typename v2::type>
```

equality operator (as type)

Template Parameters

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

8.25.2.4 inject_constant_t

```
template<typename Ring , typename X >
template<auto x>
using aerobus::Quotient< Ring, X >::inject_constant_t = val<typename Ring::template inject_constant_t<x> >
```

inject a 'constant' in quotient ring*

Template Parameters

x a 'constant' from Ring point of view

8.25.2.5 inject_ring_t

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

projects a value of Ring onto the quotient

Template Parameters

```
v a value in Ring
```

8.25.2.6 mod_t

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

modulus operator

Template Parameters

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

8.25.2.7 mul_t

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

substraction operator

Template Parameters

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

8.25.2.8 one

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

one

8.25.2.9 pos_t

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

positivity operator always true

Template Parameters

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

8.25.2.10 zero

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

zero value

8.25.3 Member Data Documentation

8.25.3.1 eq_v

```
template<typename Ring , typename X >
template<typename v1 , typename v2 >
constexpr bool aerobus::Quotient< Ring, X >::eq_v = Ring::template eq_t<typename v1::type,
typename v2::type>::value [static], [constexpr]
```

addition operator (as boolean value)

Template Parameters

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

8.25.3.2 is_euclidean_domain

```
template<typename Ring , typename X >
constexpr bool aerobus::Quotient< Ring, X >::is_euclidean_domain = true [static], [constexpr]
quotien rings are euclidean domain
```

8.25.3.3 pos_v

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

Template Parameters

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

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

· src/aerobus.h

8.26 aerobus::type_list< Ts >::split< index > Struct Template Reference

```
splits list at index
```

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

Public Types

- using head = typename inner::head
- using tail = typename inner::tail

8.26.1 Detailed Description

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

8.26.2 Member Typedef Documentation

8.26.2.1 head

index

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

8.26.2.2 tail

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

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

• src/aerobus.h

8.27 aerobus::type_list< Ts > Struct Template Reference

Empty pure template struct to handle type list.

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

Classes

struct pop_front
 removes types from head of the list
 struct split

splits list at index

Public Types

```
    template<typename T >

  using push_front = type_list< T, Ts... >
     Adds T to front of the list.
template<size_t index>
  using at = internal::type_at_t< index, Ts... >
     returns type at index
• template<typename T >
  using push_back = type_list< Ts..., T >
     pushes T at the tail of the list

    template<typename U >

  using concat = typename concat_h< U >::type
     concatenates two list into one
• template<typename T , size_t index>
  using insert = typename internal::insert_h< index, type_list< Ts... >, T >::type
     inserts type at index
• template<size t index>
  using remove = typename internal::remove_h< index, type_list< Ts... > >::type
     removes type at index
```

Static Public Attributes

```
    static constexpr size_t length = sizeof...(Ts)
    length of list
```

8.27.1 Detailed Description

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

Empty pure template struct to handle type list.

A list of types.

Template Parameters

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

8.27.2 Member Typedef Documentation

8.27.2.1 at

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

returns type at index

Template Parameters



8.27.2.2 concat

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

concatenates two list into one

Template Parameters



8.27.2.3 insert

```
template<typename... Ts>
template<typename T , size_t index>
using aerobus::type_list< Ts >::insert = typename internal::insert_h<index, type_list<Ts...>,
T>::type
```

inserts type at index

Template Parameters

index	
T	

8.27.2.4 push_back

```
template<typename... Ts>
template<typename T >
using aerobus::type_list< Ts >::push_back = type_list<Ts..., T>
pushes T at the tail of the list
Template Parameters
T
```

8.27.2.5 push_front

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

Adds T to front of the list.

Template Parameters



8.27.2.6 remove

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

removes type at index

Template Parameters



8.27.3 Member Data Documentation

8.27.3.1 length

```
template<typename... Ts>
constexpr size_t aerobus::type_list< Ts >::length = sizeof...(Ts) [static], [constexpr]
```

length of list

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

• src/aerobus.h

8.28 aerobus::type_list<> Struct Reference

specialization for empty type list

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

Public Types

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

Static Public Attributes

• static constexpr size_t length = 0

8.28.1 Detailed Description

specialization for empty type list

8.28.2 Member Typedef Documentation

8.28.2.1 concat

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

8.28.2.2 insert

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

8.28.2.3 push_back

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

8.28.2.4 push_front

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

8.28.3 Member Data Documentation

8.28.3.1 length

```
constexpr size_t aerobus::type_list<>::length = 0 [static], [constexpr]
```

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

• src/aerobus.h

8.29 aerobus::i32::val < x > Struct Template Reference

```
values in i32, again represented as types
```

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

Public Types

```
    using enclosing_type = i32
        Enclosing ring type.

    using is_zero_t = std::bool_constant< x==0 >
        is value zero
```

Static Public Member Functions

```
    template<typename valueType >
    static constexpr DEVICE valueType get ()
        cast x into valueType
    static std::string to_string ()
        string representation of value
```

Static Public Attributes

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

8.29.1 Detailed Description

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

values in i32, again represented as types
```

Template Parameters

```
x an actual integer
```

8.29.2 Member Typedef Documentation

8.29.2.1 enclosing_type

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

Enclosing ring type.

8.29.2.2 is_zero_t

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

is value zero

8.29.3 Member Function Documentation

8.29.3.1 get()

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

cast x into valueType

Template Parameters

```
valueType | double for example
```

8.29.3.2 to_string()

string representation of value

8.29.4 Member Data Documentation

8.29.4.1 v

```
template<int32_t x>
constexpr int32_t aerobus::i32::val< x >::v = x [static], [constexpr]
```

actual value stored in val type

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

• src/aerobus.h

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

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

Public Types

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

Static Public Member Functions

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

Static Public Attributes

static constexpr int64_t v = x
 actual value

8.30.1 Detailed Description

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

values in i64

Template Parameters
```

```
x an actual integer
```

Examples

examples/compensated_horner.cpp.

8.30.2 Member Typedef Documentation

8.30.2.1 enclosing_type

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

8.30.2.2 inner_type

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

type of represented values

8.30.2.3 is_zero_t

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

is value zero

8.30.3 Member Function Documentation

8.30.3.1 get()

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

cast value in valueType

Template Parameters

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

8.30.3.2 to_string()

string representation

8.30.4 Member Data Documentation

8.30.4.1 v

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

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

· src/aerobus.h

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

```
values (seen as types) in polynomial ring
```

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

Public Types

```
• using ring type = Ring
     ring coefficients live in
using enclosing_type = polynomial < Ring >
     enclosing ring type
• using aN = coeffN
     heavy weight coefficient (non zero)
• using strip = val< coeffs... >
     remove largest coefficient
• using is zero t = std::bool constant<(degree==0) &&(aN::is zero t::value)>
     true_type if polynomial is constant zero
• template<size_t index>
  using coeff_at_t = typename coeff_at< index >::type
     type of coefficient at index

    template<typename x >

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

Static Public Member Functions

```
    static std::string to_string ()
        get a string representation of polynomial
    template<typename arithmeticType >
        static constexpr DEVICE INLINED arithmeticType eval (const arithmeticType &x)
        evaluates polynomial seen as a function operating on arithmeticType
    template<typename arithmeticType >
        static DEVICE INLINED arithmeticType compensated_eval (const arithmeticType &x)
```

Evaluate polynomial on x using compensated horner scheme.

Static Public Attributes

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

• static constexpr bool is_zero_v = is_zero_t::value

true if polynomial is constant zero

8.31.1 Detailed Description

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

values (seen as types) in polynomial ring

Template Parameters

coeffN	high degree coefficient
coeffs	lower degree coefficients

Examples

examples/compensated_horner.cpp.

8.31.2 Member Typedef Documentation

8.31.2.1 aN

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

heavy weight coefficient (non zero)

8.31.2.2 coeff_at_t

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

type of coefficient at index

Template Parameters

index	

8.31.2.3 enclosing_type

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

8.31.2.4 is zero t

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
using aerobus::polynomial< Ring >::val< coeffN, coeffs >::is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>
```

true type if polynomial is constant zero

8.31.2.5 ring type

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

8.31.2.6 strip

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

8.31.2.7 value_at_t

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

8.31.3 Member Function Documentation

8.31.3.1 compensated_eval()

Evaluate polynomial on x using compensated horner scheme.

This is twice as accurate as simple eval (horner) but cannot be constexpr

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

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

Template Parameters

arithmeticType float for example

Parameters



8.31.3.2 eval()

evaluates polynomial seen as a function operating on arithmeticType

Template Parameters

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

Parameters

```
x value
```

Returns

P(x)

8.31.3.3 to_string()

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

get a string representation of polynomial

Returns

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

8.31.4 Member Data Documentation

8.31.4.1 degree

```
template<typename Ring >
template<typename coeffN , typename... coeffs>
constexpr size_t aerobus::polynomial< Ring >::val< coeffN, coeffs >::degree = sizeof...(coeffs)
[static], [constexpr]
```

degree of the polynomial

8.31.4.2 is_zero_v

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

true if polynomial is constant zero

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

• src/aerobus.h

8.32 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference

projection values in the quotient ring

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

Public Types

- using raw_t = V
- using type = abs_t< typename Ring::template mod_t< V, X >>

8.32.1 Detailed Description

projection values in the quotient ring

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

8.32.2 Member Typedef Documentation

8.32.2.1 raw_t

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

8.32.2.2 type

```
template<typename Ring , typename X >
template<typename V >
using aerobus::Quotient< Ring, X >::val< V >::type = abs_t<typename Ring::template mod_t<V,
X> >
```

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

· src/aerobus.h

8.33 aerobus::zpz::val< x > Struct Template Reference

```
values in zpz
```

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

Public Types

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

Static Public Member Functions

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

Static Public Attributes

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

8.33.1 Detailed Description

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

Template Parameters

```
x an integer
```

8.33.2 Member Typedef Documentation

8.33.2.1 enclosing_type

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

8.33.2.2 is zero t

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

true_type if zero

8.33.3 Member Function Documentation

8.33.3.1 get()

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

get value as valueType

Template Parameters

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

8.33.3.2 to_string()

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

string representation

Returns

a string representation

8.33.4 Member Data Documentation

8.33.4.1 is_zero_v

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

8.33.4.2 v

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

actual value

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

• src/aerobus.h

8.34 aerobus::polynomial< Ring >::val< coeffN > Struct Template Reference

specialization for constants

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

Classes

- struct coeff_at
- struct coeff_at< index, std::enable_if_t<(index<0||index>0)>>
- struct coeff_at< index, std::enable_if_t<(index==0)>>

Public Types

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

Static Public Member Functions

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

Static Public Attributes

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

8.34.1 Detailed Description

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

Template Parameters

```
coeffN
```

8.34.2 Member Typedef Documentation

8.34.2.1 aN

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

8.34.2.2 coeff at t

```
template<typename Ring >
template<typename coeffN >
template<size_t index>
using aerobus::polynomial< Ring >::val< coeffN >::coeff_at_t = typename coeff_at<index>
::type
```

8.34.2.3 enclosing_type

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

enclosing ring type

8.34.2.4 is_zero_t

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

8.34.2.5 ring type

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

ring coefficients live in

8.34.2.6 strip

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

8.34.2.7 value at t

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

8.34.3 Member Function Documentation

8.34.3.1 compensated eval()

8.34.3.2 eval()

8.34.3.3 to_string()

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

8.34.4 Member Data Documentation

8.34.4.1 degree

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

degree

8.34.4.2 is zero v

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

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

• src/aerobus.h

8.35 aerobus::zpz Struct Template Reference

congruence classes of integers modulo p (32 bits)

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

Classes

struct val

values in zpz

Public Types

```
• using inner_type = int32_t
     underlying type for values
template<auto x>
  using inject_constant_t = val< static_cast< int32_t >(x)>
     injects a constant integer into zpz
• using zero = val< 0 >
     zero value
• using one = val< 1 >
     one value
• template<typename v1 , typename v2 >
  using add t = typename add< v1, v2 >::type
     addition operator
• template<typename v1, typename v2 >
  using sub_t = typename sub< v1, v2 >::type
     substraction operator

    template<typename v1 , typename v2 >

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

    template<typename v1 , typename v2 >

  using div_t = typename div < v1, v2 >::type
     division operator
• template<typename v1 , typename v2 >
  using mod_t = typename remainder < v1, v2 >::type
     modulo operator
• template<typename v1 , typename v2 >
  using gt_t = typename gt < v1, v2 >::type
     strictly greater operator (type)
• template<typename v1 , typename v2 >
  using It t = typename It < v1, v2 >::type
     strictly smaller operator (type)
• template<typename v1 , typename v2 >
  using eq_t = typename eq< v1, v2 >::type
     equality operator (type)
• template<typename v1 , typename v2 >
  using gcd_t = gcd_t < i32, v1, v2 >
     greatest common divisor

    template<typename v1 >

  using pos_t = typename pos< v1 >::type
     positivity operator (type)
```

Static Public Attributes

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

8.35.1 Detailed Description

```
template < int32_t p > struct aerobus::zpz  

congruence classes of integers modulo p (32 bits) 
if p is prime, zpz 
is a field 
Template Parameters
```

Examples

a integer

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

8.35.2 Member Typedef Documentation

8.35.2.1 add_t

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

addition operator

Template Parameters

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

8.35.2.2 div_t

 $template < int32_t p >$

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

division operator

Template Parameters

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

8.35.2.3 eq_t

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

equality operator (type)

Template Parameters

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

8.35.2.4 gcd_t

```
template<int32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz::gcd_t = gcd_t<i32, v1, v2>
```

greatest common divisor

Template Parameters

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

8.35.2.5 gt_t

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

strictly greater operator (type)

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

8.35.2.6 inject_constant_t

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

injects a constant integer into zpz

Template Parameters

```
x an integer
```

8.35.2.7 inner_type

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

underlying type for values

8.35.2.8 lt_t

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

strictly smaller operator (type)

Template Parameters

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

8.35.2.9 mod_t

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

modulo operator

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

8.35.2.10 mul_t

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

multiplication operator

Template Parameters

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

8.35.2.11 one

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

one value

8.35.2.12 pos_t

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

positivity operator (type)

Template Parameters

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

8.35.2.13 sub_t

```
template<iint32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz::sub_t = typename sub<v1, v2>::type
```

substraction operator

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

8.35.2.14 zero

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

zero value

8.35.3 Member Data Documentation

8.35.3.1 eq_v

```
template<int32_t p>
template<typename v1 , typename v2 >
constexpr bool aerobus::zpz::eq_v = eq_t<v1, v2>::value [static], [constexpr]
```

equality operator (booleanvalue)

Template Parameters

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

8.35.3.2 gt_v

```
template<int32_t p>
template<typename v1 , typename v2 >
constexpr bool aerobus::zpz::gt_v = gt_t<v1, v2>::value [static], [constexpr]
```

strictly greater operator (booleanvalue)

Template Parameters

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

8.35.3.3 is_euclidean_domain

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

always true

8.35.3.4 is_field

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

true iff p is prime

8.35.3.5 lt_v

```
template<int32_t p>
template<typename v1 , typename v2 >
constexpr bool aerobus::zpz::lt_v = lt_t<v1, v2>::value [static], [constexpr]
```

strictly smaller operator (booleanvalue)

Template Parameters

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

8.35.3.6 pos_v

```
template<iint32_t p>
template<typename v >
constexpr bool aerobus::zpz::pos_v = pos_t<v>::value [static], [constexpr]
```

positivity operator (boolean value)

Template Parameters

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

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

• src/aerobus.h

Chapter 9

File Documentation

9.1 README.md File Reference

9.2 src/aerobus.h File Reference

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

9.3 aerobus.h

Go to the documentation of this file.

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

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

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

```
00215
              template<>
00216
             struct fma_helper<double> {
00217
                 static constexpr INLINED DEVICE double eval(const double x, const double y, const double
     z) {
00218
                      return x * v + z;
00219
                }
00220
             };
00221
00222
              template<>
00223
              struct fma_helper<long double> {
              static constexpr INLINED DEVICE long double eval(
00224
00225
                   const long double x, const long double y, const long double z) {
00226
                         return x * y + z;
00227
00228
             };
00229
00230
             template<>
00231
             struct fma_helper<float> {
                static constexpr INLINED DEVICE float eval(const float x, const float y, const float z) {
00232
00233
                     return x * y + z;
00234
00235
             } ;
00236
             template<>
00237
00238
             struct fma_helper<int32_t> {
                 static constexpr INLINED DEVICE int16_t eval(const int16_t x, const int16_t y, const
     int16_t z) {
00240
                      return x * y + z;
00241
00242
             };
00243
00244
              template<>
00245
             struct fma_helper<int16_t> {
00246
int32_t z) {
00247
                 static constexpr INLINED DEVICE int32_t eval(const int32_t x, const int32_t y, const
             }
                      return x * y + z;
00248
00249
             } ;
00250
00251
              template<>
00252
             struct fma_helper<int64_t> {
                 static constexpr INLINED DEVICE int64_t eval(const int64_t x, const int64_t y, const
00253
     int64 t z) {
00254
                     return x * y + z;
00255
                }
00256
             } ;
00257
             #ifdef WITH_CUDA_FP16
00258
00259
             template<>
00260
             struct fma helper< half> {
00261
                 static constexpr INLINED DEVICE _half eval(const _half x, const _half y, const _half
00262
                     #ifdef ___CUDA_ARCH_
00263
                     return __hfma(x, y, z);
00264
                     #else
00265
                     return x * y + z;
00266
                     #endif
00267
                 }
00268
00269
              template<>
             struct fma_helper<__half2> {
00270
                 static constexpr INLINED DEVICE __half2 eval(const __half2 x, const __half2 y, const
00271
__half2 z) {
                      #ifdef ___CUDA_ARCH_
00273
                      return __hfma2(x, y, z);
00274
                     #else
00275
                     return x * y + z;
00276
                     #endif
00277
                 }
00278
             } ;
              #endif
00279
         } // namespace internal
00280
00281 } // namespace aerobus
00282
00283 // compensated horner utilities
00284 namespace aerobus {
00285
       namespace internal {
00286
           template <typename T>
00287
             struct FloatLayout;
00288
             #ifdef _MSC_VER
00289
00290
             template <>
00291
              struct FloatLayout<long double> {
00292
                static constexpr uint8_t exponent = 11;
00293
                 static constexpr uint8_t mantissa = 53;
                 static constexpr uint8_t r = 27; // ceil(mantissa/2)
00294
00295
             };
```

```
00296
                #else
00297
                template <>
00298
                struct FloatLayout<long double> {
00299
                    static constexpr uint8_t exponent = 15;
                    static constexpr uint8_t mantissa = 63;
static constexpr uint8_t r = 32; // ceil(mantissa/2)
00300
00301
                    static constexpr long double shift = (1LL « r) + 1;
00302
00303
00304
                #endif
00305
00306
                template <>
                struct FloatLayout<double> {
00307
00308
                    static constexpr uint8_t exponent = 11;
00309
                    static constexpr uint8_t mantissa = 53;
00310
                     static constexpr uint8_t r = 27; // ceil(mantissa/2)
00311
                    static constexpr double shift = (1LL « r) + 1;
00312
                };
00313
00314
                template <>
00315
                struct FloatLayout<float> {
00316
                    static constexpr uint8_t exponent = 8;
00317
                     static constexpr uint8_t mantissa = 24;
                    static constant flat r = 11; // ceil(mantissa/2)
00318
00319
                    static constexpr float shift = (1 « r) + 1;
00320
00321
00322
                #ifdef WITH_CUDA_FP16
00323
                template <>
00324
                struct FloatLayout<__half> {
00325
                    static constexpr uint8_t exponent = 5;
                    static constexpr uint8_t mantissa = 11; // 10 explicitly stored
static constexpr uint8_t r = 6; // ceil(mantissa/2)
static constexpr __half shift = internal::int16_convert_helper<__half, 65>::value();
00326
00327
00328
00329
                };
00330
00331
                template <>
00332
                struct FloatLayout< half2> {
                    static constexpr uint8_t exponent = 5;
00333
00334
                     static constexpr uint8_t mantissa = 11;
                                                                     // 10 explicitely stored
                    static constexpr uint8_t r = 6; // ceil(mantissa/2)
static constexpr uint8_t r = 6; // ceil(mantissa/2)
static constexpr __half2 shift = internal::int16_convert_helper<__half2, 65>::value();
00335
00336
00337
                }:
                #endif
00338
00339
00340
                template<typename T>
00341
                static constexpr INLINED DEVICE void split(T a, T *x, T *y) {
                   T z = a * FloatLayout<T>::shift;
*x = z - (z - a);
00342
00343
                    *y = a - *x;
00344
00345
00346
00347
                template<typename T>
00348
                static constexpr INLINED DEVICE void two_sum(T a, T b, T *x, T *y) {
                  *x = a + b;
T z = *x - a;
*y = (a - (*x - z)) + (b - z);
00349
00350
00351
00352
00353
00354
                template<typename T>
                static constexpr INLINED DEVICE void two_prod(T a, T b, T *x, T *y) {
00355
00356
                    *x = a * b;
00357
                    #ifdef clang
00358
                    *y = fma_helper<T>::eval(a, b, -*x);
00359
                     #else
00360
                    T ah, al, bh, bl;
00361
                    split(a, &ah, &al);
00362
                    split(b, &bh, &bl);
*y = al * bl - (((*x - ah * bh) - al * bh) - ah * bl);
00363
                     #endif
00364
00365
                }
00366
00367
                template<typename T, size_t N>  
                static INLINED DEVICE T horner(T *p1, T *p2, T x) {
00368
                    T r = p1[0] + p2[0];

for (int64_t i = N - 1; i >= 0; --i) {

r = r * x + p1[N - i] + p2[N - i];
00369
00370
00371
00372
00373
00374
                    return r:
00375
              // namespace internal
00376
00377 } // namespace aerobus
00378
00379 // utilities
00380 namespace aerobus {
00381
           namespace internal {
00382
                template<template<typename...> typename TT, typename T>
```

```
struct is_instantiation_of : std::false_type { };
00384
00385
               template<template<typename...> typename TT, typename... Ts>
00386
               struct is_instantiation_of<TT, TT<Ts...» : std::true_type { };</pre>
00387
00388
               template<template<typename...> typename TT, typename T>
               inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00389
00390
00391
               template <int64_t i, typename T, typename... Ts>
00392
               struct type_at {
                   static_assert(i < sizeof...(Ts) + 1, "index out of range");
using type = typename type_at<i - 1, Ts...>::type;
00393
00394
00395
               };
00396
00397
               template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00398
                   using type = T;
00399
               };
00400
00401
               template <size_t i, typename... Ts>
00402
               using type_at_t = typename type_at<i, Ts...>::type;
00403
00404
00405
               template<size_t n, size_t i, typename E = void>
00406
               struct _is_prime {};
00407
00408
               template<size_t i>
00409
               struct _is_prime<0, i> {
00410
                   static constexpr bool value = false;
00411
00412
00413
               template<size t i>
00414
               struct _is_prime<1, i> {
00415
                  static constexpr bool value = false;
00416
00417
               template<size t i>
00418
00419
               struct _is_prime<2, i> {
                   static constexpr bool value = true;
00421
00422
00423
               template<size_t i>
               struct _{is\_prime<3, i> \{}
00424
                  static constexpr bool value = true;
00425
00426
00427
00428
               template<size_t i>
00429
               struct _is_prime<5, i> {
                   static constexpr bool value = true;
00430
00431
00432
00433
               template<size_t i>
00434
               struct _is_prime<7, i> {
00435
                   static constexpr bool value = true;
00436
00437
00438
               template<size_t n, size_t i>
               struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)» {
00439
00440
                   static constexpr bool value = false;
00441
00442
00443
               {\tt template} < {\tt size\_t n, size\_t i} >
               struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)» {
00444
00445
                   static constexpr bool value = false;
00446
00447
00448
               template<size_t n, size_t i>
               struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)» {
    static constexpr bool value = true;
00449
00450
00451
00452
00453
               template<size_t n, size_t i>
00454
               struct _is_prime<n, i, std::enable_if_t<(
                  n % i == 0 &&
00455
                   n >= 9 &&
00456
00457
                   n % 3 != 0 &&
00458
                   n % 2 != 0 &&
00459
                   i * i > n) \gg {
00460
                   static constexpr bool value = true;
00461
               };
00462
00463
               template<size_t n, size_t i>
               struct _is_prime<n, i, std::enable_if_t<(
    n % (i+2) == 0 &&</pre>
00464
00465
00466
                   n >= 9 &&
                   n % 3 != 0 &&
n % 2 != 0 &&
i * i <= n) » {
00467
00468
00469
```

```
static constexpr bool value = true;
00471
00472
00473
              {\tt template}{<} {\tt size\_t n, size\_t i}{\gt}
00474
              00475
00476
                      n % i != 0 &&
00477
                      n >= 9 &&
00478
                      n % 3 != 0 &&
                      n % 2 != 0 &&
00479
                      (i * i <= n))» {
00480
00481
                  static constexpr bool value = _is_prime<n, i+6>::value;
00482
00483
          } // namespace internal
00484
00487
          template<size_t n>
00488
          struct is_prime {
00490
             static constexpr bool value = internal::_is_prime<n, 5>::value;
00491
00492
00496
          template<size_t n>
00497
          static constexpr bool is_prime_v = is_prime<n>::value;
00498
00499
00500
          namespace internal {
00501
             template <std::size_t... Is>
00502
              constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
00503
                  -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
00504
00505
              template <std::size t N>
00506
              using make_index_sequence reverse
00507
                  = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00508
00514
              template<typename Ring, typename E = void>
00515
00516
00517
              template<typename Ring>
00518
              struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
00519
                  template<typename A, typename B, typename E = void>
00520
                  struct gcd_helper {};
00521
00522
                  // B = 0, A > 0
                  template<typename A, typename B>
struct gcd_helper<A, B, std::enable_if_t</pre>
00523
00524
                      ((B::is_zero_t::value) &&
00525
00526
                          (Ring::template gt_t<A, typename Ring::zero>::value))» {
00527
                      using type = A;
00528
                  };
00529
00530
                  // B = 0, A < 0
                  template<typename A, typename B>
00531
00532
                  struct gcd_helper<A, B, std::enable_if_t<
00533
                      ((B::is_zero_t::value) &&
00534
                          !(Ring::template gt_t<A, typename Ring::zero>::value))» {
00535
                      using type = typename Ring::template sub_t<typename Ring::zero, A>;
00536
                  };
00537
00538
                  // B != 0
00539
                  template<typename A, typename B>
                  struct gcd_helper<A, B, std::enable_if_t<
00540
00541
                      (!B::is_zero_t::value)
00542
                      » {
00543
                  private: // NOLINT
00544
                     // A / B
                      using k = typename Ring::template div_t<A, B>;
00545
00546
                      // A - (A/B) *B = A % B
00547
                      using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
00548
00549
                  public:
00550
                      using type = typename gcd_helper<B, m>::type;
00551
00552
00553
                  template<typename A, typename B>
00554
                  using type = typename gcd_helper<A, B>::type;
00555
00556
          } // namespace internal
00557
00558
          // vadd and vmul
00559
          namespace internal {
00560
              template<typename... vals>
00561
              struct vmul {};
00562
00563
              template<typename v1, typename... vals>
00564
              struct vmul<v1, vals...> {
00565
                 using type = typename v1::enclosing_type::template mul_t<v1, typename
     vmul<vals...>::type>;
00566
              };
```

```
template<typename v1>
00568
00569
              struct vmul<v1> {
00570
                using type = v1;
00571
00572
00573
              template<typename... vals>
00574
              struct vadd {};
00575
00576
              template<typename v1, typename... vals>
00577
             struct vadd<v1, vals...> {
                using type = typename v1::enclosing_type::template add_t<v1, typename
00578
     vadd<vals...>::type>;
00579
             };
00580
00581
              template<typename v1>
             using type = v1;
};
00582
00583
00584
00585
          } // namespace internal
00586
00589
          template<typename T, typename A, typename B>
00590
          using gcd_t = typename internal::gcd<T>::template type<A, B>;
00591
00595
          template<typename... vals>
00596
          using vadd_t = typename internal::vadd<vals...>::type;
00597
00601
          template<typename... vals>
00602
          using vmul_t = typename internal::vmul<vals...>::type;
00603
00607
          template<typename val>
00608
          requires IsEuclideanDomain<typename val::enclosing_type>
00609
          using abs_t = std::conditional_t<
00610
                          val::enclosing_type::template pos_v<val>,
00611
                          val, typename val::enclosing_type::template
      sub_t<typename val::enclosing_type::zero, val>>;
00612 } \overline{//} namespace aerobus
00613
00614 // embedding
00615 namespace aerobus {
00620
          template<typename Small, typename Large, typename E = void>
00621
          struct Embed;
00622 } // namespace aerobus
00623
00624 namespace aerobus {
00629
         template<typename Ring, typename X>
00630
          requires IsRing<Ring>
00631
          struct Quotient {
              template <typename V>
00634
              struct val {
00635
00636
              public:
00637
                  using raw_t = V;
00638
                  using type = abs_t<typename Ring::template mod_t<V, X>>;
00639
              } ;
00640
00642
              using zero = val<typename Ring::zero>;
00643
00645
              using one = val<typename Ring::one>;
00646
00650
              template<typename v1, typename v2>
              using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00651
00652
00656
              template<typename v1, typename v2>
00657
              using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00658
00662
              template<typename v1, typename v2>
00663
              using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00664
00668
              template<typename v1, typename v2>
              using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00670
00674
              template<typename v1, typename v2>
00675
              using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00676
00680
              template<typename v1, typename v2>
              static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00681
00682
00686
              template<typename v1>
00687
              using pos_t = std::true_type;
00688
00692
              template<typename v>
00693
              static constexpr bool pos_v = pos_t<v>::value;
00694
00696
              static constexpr bool is_euclidean_domain = true;
00697
00701
              template<auto x>
00702
              using inject constant t = val<typename Ring::template inject constant t<x>>:
```

```
00703
00707
               template<typename v>
00708
               using inject_ring_t = val<v>;
00709
          };
00710
00714
           template<typename Ring, typename X>
00715
          struct Embed<Quotient<Ring, X>, Ring> {
00718
               template<typename val>
00719
               using type = typename val::raw_t;
00720
00721 } // namespace aerobus
00722
00723 // type_list
00724 namespace aerobus {
00726
          template <typename... Ts>
00727
           struct type_list;
00728
00729
          namespace internal {
              template <typename T, typename... Us>
00731
               struct pop_front_h {
                   using tail = type_list<Us...>;
using head = T;
00732
00733
00734
               };
00735
00736
               template <size_t index, typename L1, typename L2>
00737
               struct split_h {
00738
00739
                   static_assert(index <= L2::length, "index ouf of bounds");</pre>
00740
                   using a = typename L2::pop_front::type;
                   using b = typename L2::pop_front::tail;
00741
00742
                   using c = typename L1::template push_back<a>;
00743
00744
00745
                   using head = typename split_h<index - 1, c, b>::head;
                   using tail = typename split_h<index - 1, c, b>::tail;
00746
00747
00748
00749
               template <typename L1, typename L2>
00750
               struct split_h<0, L1, L2> {
00751
                 using head = L1;
                   using tail = L2;
00752
00753
               };
00754
00755
               template <size_t index, typename L, typename T>
00756
               struct insert_h {
00757
                   static_assert(index <= L::length, "index ouf of bounds");</pre>
00758
                   using s = typename L::template split<index>;
00759
                   using left = typename s::head;
                   using right = typename s::tail;
00760
                   using l1 = typename left::template push_back<T>;
using type = typename l1::template concat<right>;
00761
00762
00763
00764
00765
               template <size_t index, typename L>
00766
               struct remove_h {
00767
                   using s = typename L::template split<index>;
using left = typename s::head;
00768
00769
                   using right = typename s::tail;
00770
                    using rr = typename right::pop_front::tail;
00771
                   using type = typename left::template concat<rr>;
00772
00773
           } // namespace internal
00777
           template <typename... Ts>
00778
           struct type_list {
           private:
00779
00780
               template <typename T>
00781
               struct concat h:
00782
00783
               template <typename... Us>
00784
               struct concat_h<type_list<Us...» {</pre>
00785
                   using type = type_list<Ts..., Us...>;
00786
               };
00787
00788
           public:
00790
               static constexpr size_t length = sizeof...(Ts);
00791
00794
               template <typename T>
00795
               using push_front = type_list<T, Ts...>;
00796
00799
               template <size_t index>
00800
               using at = internal::type_at_t<index, Ts...>;
00801
00803
                   using type = typename internal::pop_front_h<Ts...>::head;
using tail = typename internal::pop_front_h<Ts...>::tail;
00805
00807
00808
               };
```

```
00809
00812
              template <typename T>
00813
              using push_back = type_list<Ts..., T>;
00814
00817
              \verb|template| < \verb|typename| U >
00818
              using concat = typename concat_h<U>::type;
00819
00822
              template <size_t index>
00823
              struct split {
00824
               private:
                  using inner = internal::split_h<index, type_list<>, type_list<Ts...»;</pre>
00825
00826
00827
               public:
00828
                  using head = typename inner::head;
00829
                  using tail = typename inner::tail;
00830
00831
00835
              template <typename T, size_t index>
              using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00836
00837
00840
              template <size_t index>
00841
              using remove = typename internal::remove_h<index, type_list<Ts...»::type;</pre>
00842
          };
00843
00845
          template <>
00846
          struct type_list<> {
00847
              static constexpr size_t length = 0;
00848
00849
              template <typename T>
00850
              using push_front = type_list<T>;
00851
00852
              template <typename T>
00853
              using push_back = type_list<T>;
00854
00855
              template <typename U>
00856
              using concat = U;
00857
              // TODO(jewave): assert index == 0
00859
              template <typename T, size_t index>
00860
              using insert = type_list<T>;
00861
00862 } // namespace aerobus
00863
00864 // i16
00865 #ifdef WITH_CUDA_FP16
00866 // i16
00867 namespace aerobus {
00869
          struct i16 {
00870
             using inner_type = int16_t;
              template<int16_t x>
00873
00874
              struct val {
00876
                 using enclosing_type = i16;
00878
                  static constexpr int16_t v = x;
00879
00882
                  template<typename valueType>
00883
                  static constexpr INLINED DEVICE valueType get() {
                      return internal::template int16_convert_helper<valueType, x>::value();
00885
00886
00888
                  using is_zero_t = std::bool_constant<x == 0>;
00889
00891
                  static std::string to string() {
00892
                      return std::to_string(x);
00893
00894
              };
00895
              using zero = val<0>:
00897
00899
              using one = val<1>;
00901
              static constexpr bool is_field = false;
00903
              static constexpr bool is_euclidean_domain = true;
00906
              template<auto x>
00907
              using inject_constant_t = val<static_cast<int16_t>(x)>;
00908
00909
              template<tvpename v>
00910
              using inject_ring_t = v;
00911
00912
00913
              template<typename v1, typename v2>
00914
              struct add {
                  using type = val<v1::v + v2::v>;
00915
00916
00917
00918
              template<typename v1, typename v2>
00919
              struct sub {
00920
                  using type = val<v1::v - v2::v>;
00921
              };
00922
```

```
template<typename v1, typename v2>
00924
00925
                  using type = val<v1::v* v2::v>;
00926
00927
00928
              template<tvpename v1, tvpename v2>
00929
              struct div {
00930
                  using type = val<v1::v / v2::v>;
00931
00932
00933
              template<typename v1, typename v2>
00934
              struct remainder {
00935
                  using type = val<v1::v % v2::v>;
00936
00937
00938
              template<typename v1, typename v2>
00939
              struct qt {
00940
                 using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00941
00942
00943
              template<typename v1, typename v2>
00944
              struct lt {
                 using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00945
00946
00947
00948
              template<typename v1, typename v2>
00949
              struct eq {
00950
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00951
00952
              template<typename v1>
00953
00954
              struct pos {
00955
                  using type = std::bool_constant<(v1::v > 0)>;
00956
              };
00957
00958
           public:
              template<typename v1, typename v2>
using add_t = typename add<v1, v2>::type;
00963
00965
00970
              template<typename v1, typename v2>
00971
              using sub_t = typename sub<v1, v2>::type;
00972
00977
              template<typename v1, typename v2>
00978
              using mul_t = typename mul<v1, v2>::type;
00979
00984
              template<typename v1, typename v2>
00985
              using div_t = typename div<v1, v2>::type;
00986
00991
              template<typename v1, typename v2>
00992
              using mod t = typename remainder<v1, v2>::type;
00993
00998
              template<typename v1, typename v2>
00999
              using gt_t = typename gt<v1, v2>::type;
01000
01005
              template<typename v1, typename v2>
01006
              using lt_t = typename lt<v1, v2>::type;
01007
01012
              template<typename v1, typename v2>
01013
              using eq_t = typename eq<v1, v2>::type;
01014
01018
              template<typename v1, typename v2> \,
01019
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01020
01025
              template<typename v1, typename v2>
01026
              using gcd_t = gcd_t<i16, v1, v2>;
01027
01031
              {\tt template}{<}{\tt typename}\ {\tt v}{>}
01032
              using pos_t = typename pos<v>::type;
01033
              template<typename v>
01038
              static constexpr bool pos_v = pos_t<v>::value;
01039
01040 } // namespace aerobus
01041 #endif
01042
01043 // i32
01044 namespace aerobus {
        struct i32 {
01046
              using inner_type = int32_t;
01047
              template<int32_t x>
01050
01051
              struct val {
                  using enclosing_type = i32;
01053
01055
                  static constexpr int32_t v = x;
01056
01059
                  template<typename valueType>
                  static constexpr DEVICE valueType get() {
01060
01061
                       return static_cast<valueType>(x);
```

```
01062
                  }
01063
01065
                  using is_zero_t = std::bool_constant<x == 0>;
01066
01068
                  static std::string to string() {
01069
                      return std::to string(x);
01070
01071
01072
              using zero = val<0>;
01074
01076
              using one = val<1>;
              static constexpr bool is_field = false;
01078
01080
              static constexpr bool is_euclidean_domain = true;
01083
              template<auto x>
01084
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01085
01086
              template<typename v>
01087
              using inject_ring_t = v;
01088
01089
          private:
01090
              template<typename v1, typename v2>
01091
              struct add {
                 using type = val<v1::v + v2::v>;
01092
01093
01094
01095
              template<typename v1, typename v2>
01096
              struct sub {
01097
                 using type = val<v1::v - v2::v>;
01098
01099
01100
              template<typename v1, typename v2>
01101
              struct mul {
01102
                 using type = val<v1::v* v2::v>;
01103
              };
01104
              template<typename v1, typename v2>
01105
01106
              struct div {
01107
                  using type = val<v1::v / v2::v>;
01108
01109
01110
              template<typename v1, typename v2>
01111
              struct remainder {
                 using type = val<v1::v % v2::v>;
01112
01113
01114
01115
              template<typename v1, typename v2>
01116
              struct gt {
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01117
01118
01119
01120
              template<typename v1, typename v2>
01121
01122
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01123
01124
              template<typename v1, typename v2>
01125
01126
              struct eq {
01127
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01128
01129
              template<typename v1>
01130
01131
              struct pos {
01132
                 using type = std::bool_constant<(v1::v > 0)>;
01133
01134
           public:
01135
01140
              template<typename v1, typename v2>
01141
              using add_t = typename add<v1, v2>::type;
01142
01147
              template<typename v1, typename v2>
01148
              using sub_t = typename sub<v1, v2>::type;
01149
01154
              template<typename v1, typename v2>
01155
              using mul_t = typename mul<v1, v2>::type;
01156
01161
              template<typename v1, typename v2>
01162
              using div_t = typename div<v1, v2>::type;
01163
01168
              template<typename v1, typename v2>
              using mod_t = typename remainder<v1, v2>::type;
01169
01170
01175
              template<typename v1, typename v2>
01176
              using gt_t = typename gt<v1, v2>::type;
01177
01182
              template<typename v1, typename v2> ^{\circ}
01183
              using lt_t = typename lt<v1, v2>::type;
01184
```

```
01189
              template<typename v1, typename v2>
01190
              using eq_t = typename eq<v1, v2>::type;
01191
01195
              template<typename v1, typename v2> \,
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01196
01197
01202
              template<typename v1, typename v2>
01203
              using gcd_t = gcd_t < i32, v1, v2>;
01204
01208
              {\tt template}{<}{\tt typename}\ {\tt v}{>}
01209
              using pos_t = typename pos<v>::type;
01210
01214
              template<typename v>
01215
              static constexpr bool pos_v = pos_t < v > :: value;
01216
01217 } // namespace aerobus
01218
01219 // i64
01220 namespace aerobus {
01222
         struct i64 {
01224
             using inner_type = int64_t;
01227
              template < int64_t x >
01228
              struct val {
                  using inner_type = int32_t;
01230
01232
                  using enclosing_type = i64;
01234
                  static constexpr int64_t v = x;
01235
01238
                  template<typename valueType>
                  static constexpr INLINED DEVICE valueType get() {
01239
01240
                       return static_cast<valueType>(x);
01241
01242
01244
                  using is_zero_t = std::bool_constant<x == 0>;
01245
01247
                  static std::string to_string() {
01248
                       return std::to_string(x);
01249
                  }
01250
              };
01251
01254
              template<auto x>
01255
              using inject_constant_t = val<static_cast<int64_t>(x)>;
01256
01261
              template<typename v>
01262
              using inject_ring_t = v;
01263
01265
              using zero = val<0>;
01267
              using one = val<1>;
              static constexpr bool is_field = false;
01269
01271
              static constexpr bool is_euclidean_domain = true;
01272
           private:
01274
              template<typename v1, typename v2>
01275
              struct add {
01276
                 using type = val<v1::v + v2::v>;
01277
01278
01279
              template<typename v1, typename v2>
01280
01281
                  using type = val<v1::v - v2::v>;
01282
01283
              template<typename v1, typename v2> ^{\circ}
01284
01285
              struct mul {
01286
                 using type = val<v1::v* v2::v>;
01287
              } ;
01288
01289
              template<typename v1, typename v2>
01290
              struct div {
                  using type = val<v1::v / v2::v>;
01291
01292
              };
01293
01294
              template<typename v1, typename v2>
01295
              struct remainder {
                  using type = val<v1::v% v2::v>;
01296
01297
01298
01299
              template<typename v1, typename v2>
01300
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01301
01302
01303
01304
              template<typename v1, typename v2>
01305
01306
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01307
01308
01309
              template<tvpename v1, tvpename v2>
```

```
01310
              struct eq {
01311
                 using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01312
01313
01314
              template<typename v>
01315
              struct pos {
01316
                  using type = std::bool_constant<(v::v > 0)>;
01317
01318
01319
           public:
              template<typename v1, typename v2> ^{\circ}
01323
01324
              using add_t = typename add<v1, v2>::type;
01325
01329
              template<typename v1, typename v2>
01330
              using sub_t = typename sub<v1, v2>::type;
01331
              template<typename v1, typename v2>
01335
01336
              using mul_t = typename mul<v1, v2>::type;
01337
01342
              template<typename v1, typename v2>
01343
              using div_t = typename div<v1, v2>::type;
01344
01348
              template<typename v1, typename v2> \,
01349
              using mod_t = typename remainder<v1, v2>::type;
01350
01355
              template<typename v1, typename v2>
01356
              using gt_t = typename gt<v1, v2>::type;
01357
01362
              template<typename v1, typename v2>
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01363
01364
01369
              template<typename v1, typename v2>
01370
              using lt_t = typename lt<v1, v2>::type;
01371
              template<typename v1, typename v2>
static constexpr bool lt_v = lt_t<v1, v2>::value;
01376
01377
01378
              template<typename v1, typename v2>
01384
              using eq_t = typename eq<v1, v2>::type;
01385
01390
              template<typename v1, typename v2>
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01391
01392
01397
              template<typename v1, typename v2>
01398
              using gcd_t = gcd_t<i64, v1, v2>;
01399
01403
              template<typename v>
01404
              using pos_t = typename pos<v>::type;
01405
01409
              template<tvpename v>
              static constexpr bool pos_v = pos_t<v>::value;
01410
01411
         };
01412
01414
          template<>
          struct Embed<i32, i64> {
01415
             template<typename val>
01418
              using type = i64::val<static_cast<int64_t>(val::v)>;
01420
01421 } // namespace aerobus
01422
01423 // z/pz
01424 namespace aerobus {
01430
         template<int32_t p>
01431
          struct zpz {
01433
              using inner_type = int32_t;
01434
01437
              template<int32 t x>
01438
              struct val {
01440
                 using enclosing_type = zpz;
                  static constexpr int32_t v = x % p;
01442
01443
01446
                  template<typename valueType>
                  static constexpr INLINED DEVICE valueType get() {
01447
01448
                      return static_cast<valueType>(x % p);
01449
01450
01452
                  using is_zero_t = std::bool_constant<v == 0>;
01453
                  static constexpr bool is_zero_v = v == 0;
01455
01456
                  static std::string to string() {
01459
01460
                      return std::to_string(x % p);
01461
01462
              };
01463
01466
              template<auto x>
01467
              using inject constant t = val<static cast<int32 t>(x)>;
```

```
01468
01470
              using zero = val<0>;
01471
01473
              using one = val<1>;
01474
01476
              static constexpr bool is_field = is_prime::value;
01477
01479
              static constexpr bool is_euclidean_domain = true;
01480
01481
           private:
              template<typename v1, typename v2>
01482
01483
              struct add {
01484
                 using type = val<(v1::v + v2::v) % p>;
01485
01486
01487
              template<typename v1, typename v2>
01488
              struct sub {
01489
                 using type = val<(v1::v - v2::v) % p>;
01490
01491
01492
              template<typename v1, typename v2>
01493
              struct mul {
                 using type = val<(v1::v* v2::v) % p>;
01494
01495
01496
01497
              template<typename v1, typename v2>
01498
01499
                 using type = val<(v1::v% p) / (v2::v % p)>;
01500
01501
01502
              template<typename v1, typename v2>
01503
              struct remainder {
01504
                 using type = val<(v1::v% v2::v) % p>;
01505
01506
01507
              template<typename v1, typename v2>
01508
              struct qt {
                  using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01510
01511
01512
              template<typename v1, typename v2>
01513
              struct lt {
                 using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01514
01515
01516
01517
              template<typename v1, typename v2>
01518
                  using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01519
01520
01521
01522
              template<typename v1>
01523
              struct pos {
01524
                  using type = std::bool_constant<(v1::v > 0)>;
01525
01526
01527
           public:
              template<typename v1, typename v2>
01532
              using add_t = typename add<v1, v2>::type;
01533
01537
              template<typename v1, typename v2> \,
01538
              using sub_t = typename sub<v1, v2>::type;
01539
              template<typename v1, typename v2>
01544
              using mul_t = typename mul<v1, v2>::type;
01545
01549
              template<typename v1, typename v2>
01550
              using div_t = typename div<v1, v2>::type;
01551
01555
              template<typename v1, typename v2>
              using mod_t = typename remainder<v1, v2>::type;
01557
01561
              template<typename v1, typename v2>
01562
              using gt_t = typename gt<v1, v2>::type;
01563
              template<typename v1, typename v2>
01567
01568
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01569
01573
              template<typename v1, typename v2>
01574
              using lt_t = typename lt<v1, v2>::type;
01575
01579
              template<typename v1, typename v2>
              static constexpr bool lt_v = lt_t<v1, v2>::value;
01580
01581
01585
              template<typename v1, typename v2>
01586
              using eq_t = typename eq<v1, v2>::type;
01587
01591
              template<tvpename v1, tvpename v2>
```

```
static constexpr bool eq_v = eq_t<v1, v2>::value;
01593
01597
              template<typename v1, typename v2>
01598
              using gcd_t = gcd_t < i32, v1, v2>;
01599
01602
              template<typename v1>
01603
              using pos_t = typename pos<v1>::type;
01604
01607
              template<typename v>
01608
              static constexpr bool pos_v = pos_t<v>::value;
01609
         };
01610
         template<int32_t x>
01613
01614
          struct Embed<zpz<x>, i32> {
01617
             template <typename val>
01618
              using type = i32::val<val::v>;
01619
01620 } // namespace aerobus
01621
01622 // polynomial
01623 namespace aerobus {
         // coeffN x^N + ..
01624
01629
          template<typename Ring>
01630
         requires IsEuclideanDomain<Ring>
01631
         struct polynomial {
             static constexpr bool is_field = false;
01632
01633
              static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01634
01637
              template<typename P>
              struct horner_reduction_t {
01638
01639
                  template<size_t index, size_t stop>
01640
                  struct inner {
01641
                      template<typename accum, typename x>
01642
                      using type = typename horner_reduction_t<P>::template inner<index + 1, stop>
01643
                          ::template type<
01644
                              typename Ring::template add_t<</pre>
01645
                                  typename Ring::template mul t<x, accum>,
                                  typename P::template coeff_at_t<P::degree - index>
01646
01647
01648
                  };
01649
01650
                  template<size t stop>
01651
                  struct inner<stop, stop> {
01652
                      template<typename accum, typename x>
                      using type = accum;
01653
01654
                  };
01655
              };
01656
01660
              template<typename coeffN, typename... coeffs>
01661
              struct val {
01663
                  using ring_type = Ring;
01665
                  using enclosing_type = polynomial<Ring>;
01667
                  static constexpr size_t degree = sizeof...(coeffs);
01669
                  using aN = coeffN;
01671
                  using strip = val<coeffs...>;
                  using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01673
01675
                  static constexpr bool is_zero_v = is_zero_t::value;
01676
01677
               private:
01678
                  template<size_t index, typename E = void>
01679
                  struct coeff at {};
01680
01681
                  template<size_t index>
                  struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))» {</pre>
01682
01683
                      using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01684
01685
01686
                  template<size t index>
01687
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))» {
                      using type = typename Ring::zero;
01689
01690
               public:
01691
01694
                 template<size t index>
01695
                  using coeff at t = typename coeff at<index>::type;
01696
01699
                  static std::string to_string() {
01700
                      return string_helper<coeffN, coeffs...>::func();
01701
                  }
01702
01707
                  template<typename arithmeticType>
01708
                  static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01709
                      #ifdef WITH_CUDA_FP16
01710
                      arithmeticType start;
01711
                      if constexpr (std::is_same_v<arithmeticType, __half2>) {
01712
                          start = \underline{\quad}half2(0, 0);
                      } else {
01713
```

```
01714
                           start = static_cast<arithmeticType>(0);
01715
01716
                       #else
01717
                       arithmeticType start = static_cast<arithmeticType>(0);
01718
                       #endif
01719
                       return horner evaluation<arithmeticType, val>
01720
                               ::template inner<0, degree + 1>
01721
                               ::func(start, x);
01722
                   }
01723
01736
                  template<typename arithmeticType>
                   static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01737
01738
                       return compensated_horner<arithmeticType, val>::func(x);
01739
01740
01741
                   template<typename x>
                   using value_at_t = horner_reduction_t<val>
01742
01743
                       ::template inner<0, degree + 1>
                       ::template type<typename Ring::zero, x>;
01745
              };
01746
01749
              template<typename coeffN>
01750
               struct val<coeffN> {
                  using ring_type = Ring;
01752
                  using enclosing_type = polynomial<Ring>;
static constexpr size_t degree = 0;
01754
01756
01757
                   using aN = coeffN;
01758
                   using strip = val<coeffN>;
01759
                  using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01760
01761
                  static constexpr bool is zero v = is zero t::value;
01762
01763
                   template<size_t index, typename E = void>
01764
                   struct coeff_at {};
01765
01766
                   template<size_t index>
01767
                   struct coeff_at<index, std::enable_if_t<(index == 0)» {</pre>
01768
                       using type = aN;
01769
01770
01771
                   template<size_t index>
01772
                   struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01773
                      using type = typename Ring::zero;
01774
01775
                   template<size_t index>
01776
01777
                   using coeff_at_t = typename coeff_at<index>::type;
01778
01779
                   static std::string to_string() {
01780
                       return string_helper<coeffN>::func();
01781
                   }
01782
01783
                   template<typename arithmeticType>
01784
                   static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01785
                       return coeffN::template get<arithmeticType>();
01786
                  }
01787
01788
                   template<typename arithmeticType>
01789
                   static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01790
                       return coeffN::template get<arithmeticType>();
01791
01792
01793
                   template<typename x>
01794
                  using value_at_t = coeffN;
01795
              };
01796
01798
              using zero = val<typename Ring::zero>;
using one = val<typename Ring::one>;
01800
01802
              using X = val<typename Ring::one, typename Ring::zero>;
01803
01804
01805
              template<typename P, typename E = void>
01806
              struct simplify;
01807
01808
              template <typename P1, typename P2, typename I>
              struct add_low;
01809
01810
              template<typename P1, typename P2>
01811
01812
               struct add {
01813
                  using type = typename simplify<typename add low<
01814
                   P1,
                  P2,
01815
01816
                   internal::make_index_sequence_reverse<</pre>
01817
                   std::max(P1::degree, P2::degree) + 1
01818
                   »::type>::type;
01819
              };
01820
```

```
template <typename P1, typename P2, typename I>
01822
01823
01824
              template <typename P1, typename P2, typename I>
01825
              struct mul low;
01826
01827
              template<typename v1, typename v2>
01828
              struct mul
01829
                      using type = typename mul_low<
01830
01831
                          v2.
01832
                          internal::make_index_sequence_reverse<
01833
                          v1::degree + v2::degree + 1
01834
                          »::type;
01835
              };
01836
01837
              template<typename coeff, size_t deg>
01838
              struct monomial;
01839
01840
              template<typename v, typename E = void>
01841
              struct derive_helper {};
01842
01843
              template<typename v>
01844
              struct derive_helper<v, std::enable_if_t<v::degree == 0» {</pre>
01845
                  using type = zero;
01846
01847
01848
              template<typename v>
01849
              struct derive_helper<v, std::enable_if_t<v::degree != 0» {
01850
                  using type = typename add<
01851
                      typename derive_helper<typename simplify<typename v::strip>::type>::type,
01852
                      typename monomial<
01853
                           typename Ring::template mul_t<</pre>
01854
                               typename v::aN,
01855
                              typename Ring::template inject_constant_t<(v::degree)>
01856
01857
                          v::degree - 1
01858
                      >::type
01859
                  >::type;
01860
01861
01862
              template<typename v1, typename v2, typename E = void>
01863
              struct eq helper {};
01864
              01865
01866
01867
                  using type = std::false_type;
01868
              };
01869
01870
01871
              template<typename v1, typename v2>
01872
              struct eq_helper<v1, v2, std::enable_if_t<
01873
                  v1::degree == v2::degree &&
                  (v1::degree != 0 || v2::degree != 0) &&
01874
01875
                  std::is_same<
01876
                  typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01877
                  std::false_type
01878
                  >::value
01879
01880
              > {
01881
                  using type = std::false type;
01882
              };
01883
01884
              template<typename v1, typename v2>
struct eq_helper<v1, v2, std::enable_if_t<</pre>
01885
                  v1::degree == v2::degree &&
(v1::degree != 0 || v2::degree != 0) &&
01886
01887
01888
                  std::is same<
01889
                  typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
                  std::true_type
01891
01892
              » {
01893
                  using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01894
              };
01895
01896
              template<typename v1, typename v2>
01897
              struct eq_helper<v1, v2, std::enable_if_t<
01898
                  v1::degree == v2::degree &&
                  (v1::degree == 0)
01899
01900
              » {
01901
                  using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01902
              };
01903
01904
              template<typename v1, typename v2, typename E = void>
01905
              struct lt_helper {};
01906
01907
              template<tvpename v1, tvpename v2>
```

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

```
using sub_at_t = typename sub_at<P1, P2, index>::type;
01996
01997
               template<typename P1, typename P2, std::size_t... I>
01998
              struct sub_low<P1, P2, std::index_sequence<I...» {
01999
                  using type = val<sub_at_t<P1, P2, I>...>;
02000
02001
02002
               template<typename P1, typename P2>
               struct sub {
02003
02004
                   using type = typename simplify<typename sub_low<
02005
                  P1.
02006
                  P2.
02007
                   internal::make_index_sequence_reverse<
02008
                   std::max(P1::degree, P2::degree) + 1
02009
                   »::type>::type;
02010
02011
               // multiplication at
02012
02013
               template<typename v1, typename v2, size_t k, size_t index, size_t stop>
02014
              struct mul_at_loop_helper {
                   using type = typename Ring::template add_t<
02015
02016
                       typename Ring::template mul_t<</pre>
                       typename v1::template coeff_at_t<index>,
02017
02018
                       typename v2::template coeff_at_t<k - index>
02019
02020
                       typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
02021
02022
              };
02023
              template<typename v1, typename v2, size_t k, size_t stop> struct mul_at_loop_helper<v1, v2, k, stop, stop> {
02024
02025
02026
                  using type = typename Ring::template mul_t<
02027
                       typename v1::template coeff_at_t<stop>,
02028
                       typename v2::template coeff_at_t<0>;
02029
02030
02031
              template <typename v1, typename v2, size_t k, typename E = void>
02032
              struct mul_at {};
02033
              02034
02035
02036
                  using type = typename Ring::zero;
02037
02038
              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)» {
02039
02040
02041
                  using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
02042
              };
02043
              template<typename P1, typename P2, size_t index>
02044
02045
              using mul_at_t = typename mul_at<P1, P2, index>::type;
02046
02047
               template<typename P1, typename P2, std::size_t... I>
02048
               struct mul_low<P1, P2, std::index_sequence<I...» {
02049
                   using type = val<mul_at_t<P1, P2, I>...>;
02050
              };
02051
02052
               // division helper
02053
               template< typename A, typename B, typename Q, typename R, typename E = void>
02054
               struct div_helper {};
02055
              template<typename A, typename B, typename Q, typename R>
struct div_helper<A, B, Q, R, std::enable_if_t<</pre>
02056
02057
02058
                  (R::degree < B::degree) ||
02059
                   (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
02060
                  using q_type = Q;
02061
                  using mod_type = R;
02062
                  using gcd_type = B;
02063
02064
02065
               template<typename A, typename B, typename Q, typename R>
02066
               struct div_helper<A, B, Q, R, std::enable_if_t<
02067
                   (R::degree >= B::degree) &&
                   !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
02068
               private: // NOLINT
    using rN = typename R::aN;
02069
02070
02071
                   using bN = typename B::aN;
02072
                   using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
     B::degree>::type;
02073
                  using rr = typename sub<R, typename mul<pT, B>::type>::type;
                  using qq = typename add<Q, pT>::type;
02074
02075
02076
               public:
                  using q_type = typename div_helper<A, B, qq, rr>::q_type;
02077
                  using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
using gcd_type = rr;
02078
02079
02080
              };
```

```
02081
02082
               template<typename A, typename B>
02083
               struct div {
02084
                  static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
02085
                  using q_type = typename div_helper<A, B, zero, A>::q_type;
using m_type = typename div_helper<A, B, zero, A>::mod_type;
02086
02087
02088
               template<typename P>
02089
02090
               struct make unit {
02091
                   using type = typename div<P, val<typename P::aN»::q_type;
02092
02093
02094
               template<typename coeff, size_t deg>
02095
               struct monomial {
02096
                  using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
02097
              };
02098
02099
               template<typename coeff>
02100
              struct monomial<coeff, 0>
                  using type = val<coeff>;
02101
02102
02103
               template<typename arithmeticType, typename P>
02104
02105
               struct horner_evaluation {
02106
                 template<size_t index, size_t stop>
02107
                   struct inner {
02108
                       static constexpr DEVICE INLINED arithmeticType func(
                           const arithmeticType& accum, const arithmeticType& x) {
return horner_evaluation<arithmeticType, P>::template inner<index + 1,</pre>
02109
02110
      stop>::func(
02111
                                internal::fma_helper<arithmeticType>::eval(
02112
                                    х,
02113
                                    accum,
02114
                                    P::template coeff_at_t<P::degree - index>::template
      get<arithmeticType>()), x);
02115
02116
                  };
02117
02118
                  template<size_t stop>
02119
                   struct inner<stop, stop> {
                       static constexpr DEVICE INLINED arithmeticType func(
02120
02121
                          const arithmeticType& accum, const arithmeticType& x) {
02122
                           return accum;
02123
02124
                   };
02125
              };
02126
               template<typename arithmeticType, typename P>
02127
02128
               struct compensated horner {
                   template<int64_t index, int ghost>
02130
                   struct EFTHorner {
02131
                       static INLINED void func(
02132
                               arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType
      *r) {
02133
                           arithmeticType p;
02134
                           internal::two_prod(*r, x, &p, pi + P::degree - index - 1);
                            constexpr arithmeticType coeff = P::template coeff_at_t<index>::template
02135
      get<arithmeticType>();
02136
                           internal::two_sum<arithmeticType>(
                               p, coeff,
r, sigma + P::degree - index - 1);
02137
02138
02139
                           EFTHorner<index - 1, ghost>::func(x, pi, sigma, r);
02140
02141
                   };
02142
02143
                   template<int ghost>
                   struct EFTHorner<-1, ghost> {
02144
02145
                       static INLINED DEVICE void func(
02146
                               arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType
      *r) {
02147
02148
                   };
02149
                   static INLINED DEVICE arithmeticType func(arithmeticType x) {
02150
02151
                      arithmeticType pi[P::degree], sigma[P::degree];
02152
                       arithmeticType r = P::template coeff_at_t<P::degree>::template get<arithmeticType>();
02153
                       EFTHorner<P::degree - 1, 0>::func(x, pi, sigma, &r);
02154
                       arithmeticType c = internal::horner<arithmeticType, P::degree - 1>(pi, sigma, x);
02155
                       return r + c;
02156
                   }
02157
               } ;
02158
02159
               template<typename coeff, typename... coeffs>
02160
               struct string_helper {
02161
                   static std::string func() {
02162
                       std::string tail = string helper<coeffs...>::func();
```

```
02163
                       std::string result = "";
02164
                       if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
02165
                           return tail;
                       } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
02166
02167
                          if (sizeof...(coeffs) == 1) {
    result += "x";
02168
                           } else {
02169
02170
                               result += "x^" + std::to_string(sizeof...(coeffs));
02171
02172
                       } else {
                           if (sizeof...(coeffs) == 1) {
02173
                               result += coeff::to_string() + " x";
02174
02175
                           } else {
02176
                               result += coeff::to_string()
02177
                                        + " x^" + std::to_string(sizeof...(coeffs));
02178
02179
02180
02181
                       if (!tail.empty()) {
                          if (tail.at(0) != '-') {
    result += " + " + tail;
02182
02183
02184
                           } else {
                               result += " - " + tail.substr(1);
02185
02186
02187
02188
02189
                       return result;
02190
                  }
02191
              };
02192
02193
              template<typename coeff>
02194
              struct string_helper<coeff>
02195
                  static std::string func() {
02196
                      if (!std::is_same<coeff, typename Ring::zero>::value) {
                      return coeff::to_string();
} else {
02197
02198
                           return "";
02199
02200
02201
                  }
02202
              };
02203
02204
           public:
              template<typename P>
02207
02208
              using simplify_t = typename simplify<P>::type;
02209
02213
              template<typename v1, typename v2>
02214
              using add_t = typename add<v1, v2>::type;
02215
02219
              template<typename v1, typename v2>
02220
              using sub t = typename sub<v1, v2>::type;
02225
              template<typename v1, typename v2>
02226
              using mul_t = typename mul<v1, v2>::type;
02227
02231
              template<typename v1, typename v2>
02232
              using eq_t = typename eq_helper<v1, v2>::type;
02237
              template<typename v1, typename v2>
02238
              using lt_t = typename lt_helper<v1, v2>::type;
02239
02243
              template<typename v1, typename v2>
02244
              using gt_t = typename gt_helper<v1, v2>::type;
02245
02249
              template<typename v1, typename v2>
02250
              using div_t = typename div<v1, v2>::q_type;
02251
02255
              template<typename v1, typename v2>
02256
              using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
02257
02261
              template<typename coeff, size_t deg>
02262
              using monomial_t = typename monomial<coeff, deg>::type;
02263
02266
              template<typename v>
02267
              using derive_t = typename derive_helper<v>::type;
02268
02271
              template<typename v>
02272
              using pos_t = typename Ring::template pos_t<typename v::aN>;
02273
02276
              template<typename v>
02277
              static constexpr bool pos v = pos t<v>::value;
02278
02282
              template<typename v1, typename v2>
02283
              using gcd_t = std::conditional_t<
02284
                  Ring::is_euclidean_domain,
02285
                  typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
02286
                  void>:
02287
```

```
template<auto x>
02291
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
02292
02295
              template<typename v>
              using inject_ring_t = val<v>;
02296
02297
          };
02298 } // namespace aerobus
02299
02300 // fraction field
02301 namespace aerobus {
02302
          namespace internal {
             template<typename Ring, typename E = void>
02303
02304
              requires IsEuclideanDomain<Ring>
02305
              struct _FractionField {};
02306
02307
              template<typename Ring>
              requires IsEuclideanDomain<Ring>
02308
              struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain> {
    static constexpr bool is_field = true;
02309
02311
02312
                  static constexpr bool is_euclidean_domain = true;
02313
02314
02315
                  template<typename val1, typename val2, typename E = void>
02316
                  struct to_string_helper {};
02317
02318
                  template<typename val1, typename val2>
02319
                   struct to_string_helper <val1, val2,
02320
                       std::enable_if_t<
02321
                       Ring::template eq_t<
02322
                       val2, typename Ring::one
02323
                      >::value
02324
02325
02326
                       static std::string func() {
02327
                          return vall::to_string();
02328
02329
                  };
02330
02331
                   template<typename val1, typename val2>
02332
                   struct to_string_helper<val1, val2,
02333
                       std::enable_if_t<
02334
                       !Ring::template eq_t<
02335
                       val2.
02336
                       typename Ring::one
02337
                       >::value
02338
                       >
02339
02340
                       static std::string func() {
                           return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
02341
02342
02343
                  };
02344
02345
               public:
02349
                  template<typename val1, typename val2>
02350
                   struct val {
02352
                      using x = val1;
02354
                      using y = val2;
02356
                      using is_zero_t = typename val1::is_zero_t;
02358
                      static constexpr bool is_zero_v = val1::is_zero_t::value;
02359
02361
                      using ring_type = Ring;
                      using enclosing_type = _FractionField<Ring>;
02362
02363
02366
                       static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
02367
02368
                       template<typename valueType, int ghost = 0>
02369
                       struct get_helper {
                          static constexpr INLINED DEVICE valueType get() {
02370
02371
                               return internal::staticcast<valueType, typename
      ring_type::inner_type>::template func<x::v>() /
02372
                                   internal::staticcast<valueType, typename ring_type::inner_type>::template
      func<y::v>();
02373
02374
                      };
02375
02376
                       #ifdef WITH_CUDA_FP16
02377
                       template<int ghost>
02378
                       struct get_helper<__half, ghost> {
                           static constexpr INLINED DEVICE
02379
                                                              _half get() {
                               return internal::my_float2half_rn(
02380
                                   internal::staticcast<float, typename ring_type::inner_type>::template
02381
      func<x::v>() /
02382
                                   internal::staticcast<float, typename ring_type::inner_type>::template
      func<y::v>());
02383
                           }
02384
                      };
02385
```

```
template<int ghost>
                      struct get_helper<__half2, ghost> {
02387
                          static constexpr INLINED DEVICE __half2 get() {
02388
                              constexpr __half tmp = internal::my_float2half_rn(
02389
02390
                                  internal::staticcast<float, typename ring_type::inner_type>::template
      func<x::v>() /
02391
                                  internal::staticcast<float, typename ring_type::inner_type>::template
      func<y::v>());
02392
                              return __half2(tmp, tmp);
02393
                          }
02394
                      };
02395
                      #endif
02396
02400
                      template<typename valueType>
02401
                      static constexpr INLINED DEVICE valueType get() {
02402
                          return get_helper<valueType, 0>::get();
02403
02404
02407
                      static std::string to_string() {
02408
                          return to_string_helper<val1, val2>::func();
02409
02410
02415
                      template<typename arithmeticType>
02416
                      static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& v) {
02417
                          return x::eval(v) / y::eval(v);
02418
02419
                  };
02420
02422
                  using zero = val<typename Ring::zero, typename Ring::one>;
02424
                  using one = val<typename Ring::one, typename Ring::one>;
02425
02428
                  template<typename v>
02429
                  using inject_t = val<v, typename Ring::one>;
02430
02433
                  template<auto x>
                  using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
02434
     Ring::one>;
02435
02438
                  template<typename v>
02439
                  using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
02440
02442
                  using ring type = Ring;
02443
02444
               private:
                  template<typename v, typename E = void>
02445
02446
                  struct simplify {};
02447
                  // x = 0
02448
                  template<tvpename v>
02449
02450
                  struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {
02451
                      using type = typename _FractionField<Ring>::zero;
02452
02453
02454
                  // x != 0
02455
                  template<typename v>
                  struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value» {</pre>
02456
02457
02458
                      using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
02459
                      using newx = typename Ring::template div_t<typename v::x, _gcd>;
02460
                      using newy = typename Ring::template div_t<typename v::y, _gcd>;
02461
02462
                      using posx = std::conditional t<
02463
                                           !Ring::template pos_v<newy>,
02464
                                           typename Ring::template sub_t<typename Ring::zero, newx>,
02465
                                           newx>;
02466
                      using posy = std::conditional_t<
02467
                                           !Ring::template pos_v<newy>,
02468
                                           typename Ring::template sub_t<typename Ring::zero, newy>,
02469
                                           newv>;
02470
                   public:
02471
                      using type = typename _FractionField<Ring>::template val<posx, posy>;
02472
                  };
02473
02474
               public:
02477
                  template<typename v>
02478
                  using simplify_t = typename simplify<v>::type;
02479
02480
               private:
02481
                  template<typename v1, typename v2>
02482
                  struct add {
02483
                   private:
02484
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02485
                      using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02486
                      using dividend = typename Ring::template add_t<a, b>;
02487
                      using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02488
                      using g = typename Ring::template gcd_t<dividend, diviser>;
02489
```

```
02490
                   public:
                       using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
      diviser»;
02492
02493
02494
                   template<tvpename v>
02495
                   struct pos {
02496
                       using type = std::conditional_t<
02497
                            (\texttt{Ring::template pos\_v<typename v::x> \&\& Ring::template pos\_v<typename v::y>) \ |\ |
02498
                            (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
02499
                           std::true_type,
02500
                           std::false type>;
02501
                  };
02502
02503
                   template<typename v1, typename v2>
02504
                   struct sub {
                    private:
02505
02506
                       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>;
02508
                       using dividend = typename Ring::template sub_t<a, b>;
02509
                       using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02510
                       using g = typename Ring::template gcd_t<dividend, diviser>;
02511
                    public:
02512
02513
                       using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
     diviser»;
02514
02515
02516
                   template<typename v1, typename v2>
02517
                   struct mul {
02518
                    private:
02519
                       using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
02520
                       using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02521
02522
                   public:
                       using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;</pre>
02523
02524
                   };
02526
                   template<typename v1, typename v2, typename E = void>
02527
                   struct div {};
02528
02529
                   template<typename v1, typename v2> \,
struct div<v1, v2, std::
_FractionField<Ring>::zero>::value>
02531
                   struct div<v1, v2, std::enable if t<!std::is same<v2, typename
                   private:
02532
                       using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02533
                       using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02534
02535
                   public:
02536
                      using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
                   };
02538
02539
                   template<typename v1, typename v2>
02540
                   struct div<v1, v2, std::enable_if_t<
02541
                       std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
02542
                       using type = one;
02543
02544
02545
                   template<typename v1, typename v2>
                   struct eq {
02546
02547
                       using type = std::conditional t<
                               std::is_same<typename simplify_t<vl>::x, typename simplify_t<v2>::x>::value &&
02548
02549
                                std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
02550
                           std::true_type,
02551
                           std::false_type>;
02552
                   };
02553
02554
                   template<typename v1, typename v2, typename E = void>
02555
                   struct at:
02557
                   template<typename v1, typename v2>
02558
                   struct gt<v1, v2, std::enable_if_t<
02559
                       (eq<v1, v2>::type::value)
02560
02561
                       using type = std::false type;
02562
02563
02564
                   template<typename v1, typename v2>
                   struct gt<v1, v2, std::enable_if_t<
    (!eq<v1, v2>::type::value) &&
    (!pos<v1>::type::value) && (!pos<v2>::type::value)
02565
02566
02567
02568
02569
                       using type = typename gt<
02570
                           typename sub<zero, v1>::type, typename sub<zero, v2>::type
02571
                       >::type;
02572
                   };
02573
```

```
template<typename v1, typename v2>
                   struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02575
02576
                        (pos<v1>::type::value) && (!pos<v2>::type::value)
02577
02578
02579
                        using type = std::true type;
02580
                   };
02581
02582
                   template<typename v1, typename v2>
                   struct gt<v1, v2, std::enable_if_t<
    (!eq<v1, v2>::type::value) &&
    (!pos<v1>::type::value) && (pos<v2>::type::value)
02583
02584
02585
02586
02587
                        using type = std::false_type;
02588
                   };
02589
02590
                   template<typename v1, typename v2>
                   struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
02591
02592
02593
                        (pos<v1>::type::value) && (pos<v2>::type::value)
02594
02595
                        using type = typename Ring::template gt_t<
02596
                            typename Ring::template mul_t<v1::x, v2::y>,
02597
                            typename Ring::template mul_t<v2::y, v2::x>
02598
02599
                   };
02600
                public:
02601
02605
                   template<typename v1, typename v2>
02606
                   using add_t = typename add<v1, v2>::type;
02607
02612
                   template<typename v1, typename v2>
02613
                   using mod_t = zero;
02614
02619
                   template<typename v1, typename v2>
02620
                   using gcd_t = v1;
02621
02625
                   template<typename v1, typename v2>
02626
                   using sub_t = typename sub<v1, v2>::type;
02627
02631
                   template<typename v1, typename v2>
02632
                   using mul_t = typename mul<v1, v2>::type;
02633
02637
                   template<typename v1, typename v2>
02638
                   using div_t = typename div<v1, v2>::type;
02639
02643
                   template<typename v1, typename v2>
02644
                   using eq_t = typename eq<v1, v2>::type;
02645
02649
                   template<tvpename v1, tvpename v2>
02650
                   static constexpr bool eq_v = eq<v1, v2>::type::value;
02651
02655
                   template<typename v1, typename v2>
02656
                   using gt_t = typename gt<v1, v2>::type;
02657
02661
                   template<typename v1, typename v2>
02662
                   static constexpr bool gt_v = gt<v1, v2>::type::value;
02663
02666
                   template<typename v1>
02667
                   using pos_t = typename pos<v1>::type;
02668
02671
                   template<typename v>
02672
                    static constexpr bool pos_v = pos_t<v>::value;
02673
02674
02675
               template<typename Ring, typename E = void>
02676
               requires IsEuclideanDomain<Ring>
               struct FractionFieldImpl {};
02677
02678
               // fraction field of a field is the field itself
02680
               template<typename Field>
02681
               requires IsEuclideanDomain<Field>
02682
               struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {</pre>
02683
                   using type = Field;
02684
                   template<typename v>
02685
                   using inject_t = v;
02686
02687
               \ensuremath{//} fraction field of a ring is the actual fraction field
02688
02689
               template<typename Ring>
               requires IsEuclideanDomain<Ring>
02690
               struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {
   using type = _FractionField<Ring>;
02691
02692
02693
               };
02694
           } // namespace internal
02695
02698
          template<tvpename Ring>
```

```
requires IsEuclideanDomain<Ring>
          using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02700
02701
02704
          template<typename Ring>
          struct Embed<Ring, FractionField<Ring> {
02705
02708
              template<tvpename v>
02709
              using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02710
02711 } // namespace aerobus
02712
02713
02714 // short names for common types
02715 namespace aerobus {
02719
          template<typename X, typename Y>
02720
          requires IsRing<typename X::enclosing_type> &&
02721
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02722
          using add_t = typename X::enclosing_type::template add_t<X, Y>;
02723
          template<typename X, typename Y>
02728
          requires IsRing<typename X::enclosing_type> &&
02729
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02730
          using sub_t = typename X::enclosing_type::template sub_t<X, Y>;
02731
02735
          template<typename X, typename Y>
requires IsRing<typename X::enclosing_type> &&
02736
02737
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02738
          using mul_t = typename X::enclosing_type::template mul_t<X, Y>;
02739
02743
          template<typename X, typename Y>
02744
          requires IsEuclideanDomain<typename X::enclosing_type> &&
02745
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02746
          using div_t = typename X::enclosing_type::template div_t<X, Y>;
02747
02750
          using q32 = FractionField<i32>;
02751
02754
          using fpq32 = FractionField<polynomial<q32>>;
02755
02758
          using q64 = FractionField<i64>;
02759
02761
          using pi64 = polynomial<i64>;
02762
02764
          using pg64 = polynomial<g64>;
02765
02767
          using fpq64 = FractionField<polynomial<q64>>;
02768
02773
          template<typename Ring, typename v1, typename v2>
02774
          using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02775
02782
          template<tvpename v>
02783
          using embed_int_poly_in_fractions_t =
02784
                  typename Embed<
02785
                      polynomial<typename v::ring_type>,
02786
                       polynomial<FractionField<typename v::ring_type>>>::template type<v>;
02787
          template<int64_t p, int64_t q>
using make_q64_t = typename q64::template simplify_t<</pre>
02791
02792
02793
                       typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02794
          template<int32_t p, int32_t q>
using make_q32_t = typename q32::template simplify_t<</pre>
02798
02799
                       typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>»;
02800
02801
02806
          template<typename Ring, typename v1, typename v2>
          using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02807
02812
          template<typename Ring, typename v1, typename v2>
02813
          using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02814
02816
          template<>
02817
          struct Embed<q32, q64> {
02820
              template<typename v>
02821
              using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02822
02823
02827
          template<typename Small, typename Large>
02828
          struct Embed<polynomial<Small>, polynomial<Large» {</pre>
          private:
02829
02830
              template<typename v, typename i>
02831
              struct at_low;
02832
02833
              template<typename v, size t i>
02834
              struct at_index {
02835
                  using type = typename Embed<Small, Large>::template
      type<typename v::template coeff_at_t<i>>;
02836
02837
02838
              template<typename v, size_t... Is>
02839
              struct at_low<v, std::index_sequence<Is...» {</pre>
```

```
using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
02841
             };
02842
          public:
02843
02846
             template<typename v>
02847
             using type = typename at low<v, typename internal::make index sequence reverse<v::degree +
     1»::type;
02848
02849
02853
         template<typename Ring, auto... xs>
         using make_int_polynomial_t = typename polynomial<Ring>::template val<</pre>
02854
02855
                 typename Ring::template inject_constant_t<xs>...>;
02856
02860
         template<typename Ring, auto... xs>
02861
         using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<
02862
                 typename FractionField<Ring>::template inject_constant_t<xs>...>;
02863 } // namespace aerobus
02864
02865 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02866 namespace aerobus {
         namespace internal {
02867
02868
             template<typename T, size_t x, typename E = void>
02869
             struct factorial {};
02870
02871
             template<typename T, size_t x>
             struct factorial<T, x, std::enable_if_t<(x > 0)» {
02872
02873
             private:
02874
                 template<typename, size_t, typename>
02875
                 friend struct factorial;
02876
             public:
                 using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
02877
     x - 1>::tvpe>;
02878
                 static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02879
02880
             template<typename T>
02881
             struct factorial<T, 0> {
02883
              public:
02884
                using type = typename T::one;
02885
                 static constexpr typename T::inner_type value = type::template get<typename</pre>
     T::inner_type>();
02886
             };
02887
         } // namespace internal
02888
02892
         template<typename T, size_t i>
02893
         using factorial_t = typename internal::factorial<T, i>::type;
02894
02898
         template<tvpename T, size t i>
02899
         inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02900
02901
         namespace internal {
02902
             template<typename T, size_t k, size_t n, typename E = void>
02903
              struct combination_helper {};
02904
02905
             template<typename T, size t k, size t n>
02906
             struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
02907
                 using type = typename FractionField<T>::template mul_t<</pre>
02908
                     typename combination_helper<T, k - 1, n - 1>::type,
02909
                     makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02910
             };
02911
02912
             template<typename T, size_t k, size_t n>
02913
             struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)   {
02914
                 using type = typename combination_helper<T, n - k, n>::type;
02915
02916
02917
             template<typename T, size_t n>
02918
             struct combination_helper<T, 0, n> {
                 using type = typename FractionField<T>::one;
02920
02921
02922
             template<typename T, size_t k, size_t n>
02923
             struct combination {
02924
                 using type = typename internal::combination helper<T, k, n>::type::x;
02925
                 static constexpr typename T::inner_type value =
                             internal::combination_helper<T, k, n>::type::template get<typename</pre>
02926
     T::inner_type>();
02927
         } // namespace internal
02928
02929
02932
         template<typename T, size_t k, size_t n>
02933
         using combination_t = typename internal::combination<T, k, n>::type;
02934
02939
         template<typename T, size_t k, size_t n>
         02940
02941
```

```
namespace internal {
02943
              template<typename T, size_t m>
02944
              struct bernoulli;
02945
02946
              template<typename T, typename accum, size_t k, size_t m>
              struct bernoulli_helper {
02947
02948
                  using type = typename bernoulli_helper<
02949
02950
                       addfractions_t<T,
02951
                           accum,
                           mulfractions_t<T,</pre>
02952
02953
                               makefraction t<T.
02954
                                   combination_t<T, k, m + 1>,
                                   typename T::one>,
02955
02956
                               typename bernoulli<T, k>::type
02957
                           >
02958
                       >,
k + 1,
02959
02960
                      m>::type;
02961
              };
02962
02963
              template<typename T, typename accum, size_t m>
02964
              struct bernoulli_helper<T, accum, m, m> {
02965
                  using type = accum;
02966
02967
02968
02969
02970
              template<typename T, size_t m>
02971
              struct bernoulli {
02972
                  using type = typename FractionField<T>::template mul_t<</pre>
                       typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02974
02975
                       typename T::template val<static_cast<typename T::inner_type>(-1)>,
02976
                       typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02977
02978
                  >;
02979
02980
                  template<typename floatType>
02981
                  static constexpr floatType value = type::template get<floatType>();
02982
              };
02983
02984
              template<typename T>
02985
              struct bernoulli<T, 0> {
02986
                  using type = typename FractionField<T>::one;
02987
02988
                  template<typename floatType>
02989
                  static constexpr floatType value = type::template get<floatType>();
02990
              };
02991
          } // namespace internal
02992
02996
          template<typename T, size_t n>
02997
          using bernoulli_t = typename internal::bernoulli<T, n>::type;
02998
          template<typename FloatType, typename T, size_t n >
inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
03003
03004
03005
03006
          // bell numbers
03007
          namespace internal {
              template<typename T, size_t n, typename E = void>
03008
03009
              struct bell_helper;
03010
03011
              template <typename T, size_t n>
03012
              struct bell_helper<T, n, std::enable_if_t<(n > 1)» {
03013
                  template<typename accum, size_t i, size_t stop>
03014
                  struct sum_helper {
03015
                   private:
03016
                      using left = typename T::template mul_t<
                                   combination_t<T, i, n-1>,
03017
                                   typename bell_helper<T, i>::type>;
03018
03019
                       using new_accum = typename T::template add_t<accum, left>;
                   public:
03020
03021
                       using type = typename sum_helper<new_accum, i+1, stop>::type;
03022
                  };
03023
03024
                   template<typename accum, size_t stop>
03025
                  struct sum_helper<accum, stop, stop> {
03026
                      using type = accum;
03027
                  };
03028
03029
                  using type = typename sum_helper<typename T::zero, 0, n>::type;
03030
              };
03031
03032
              template<typename T>
03033
              struct bell_helper<T, 0> {
03034
                  using type = typename T::one;
03035
              };
```

```
03036
              template<typename T>
03037
03038
              struct bell_helper<T, 1> {
03039
                 using type = typename T::one;
03040
03041
          } // namespace internal
03042
03046
          template<typename T, size_t n>
03047
          using bell_t = typename internal::bell_helper<T, n>::type;
03048
03052
          template<typename T, size_t n>
          static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
03053
03054
03055
          namespace internal {
03056
              template<typename T, int k, typename E = void>
03057
              struct alternate {};
03058
03059
              template<typename T, int k>
03060
              struct alternate<T, k, std::enable_if_t<k % 2 == 0» {</pre>
03061
                  using type = typename T::one;
                  static constexpr typename T::inner_type value = type::template get<typename</pre>
03062
     T::inner_type>();
03063
             };
03064
03065
              template<typename T, int k>
              struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
03066
03067
                  using type = typename T::template sub_t<typename T::zero, typename T::one>;
03068
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
03069
              };
03070
          } // namespace internal
03071
03074
          template<typename T, int k>
03075
          using alternate_t = typename internal::alternate<T, k>::type;
03076
          template<typename T, size_t k>
03079
03080
          inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
03081
03082
          namespace internal {
03083
              template<typename T, int n, int k, typename E = void>
03084
              struct stirling_1_helper {};
03085
03086
              template<typename T>
03087
              struct stirling_1_helper<T, 0, 0> {
                  using type = typename T::one;
03088
03089
03090
03091
              template<typename T, int n>
              struct stirling_1_helper<T, n, 0, std::enable_if_t<(n > 0)» {
03092
                  using type = typename T::zero;
03093
03094
              };
03095
03096
              template<typename T, int n>
              struct stirling_1_helper<T, 0, n, std::enable_if_t<(n > 0)» {
03097
03098
                  using type = typename T::zero;
03099
              };
03100
03101
              template<typename T, int n, int k>
03102
              struct stirling_1_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)» {
03103
                  using type = typename T::template sub_t<
                                   typename stirling_1_helper<T, n-1, k-1>::type,
typename T::template mul_t<</pre>
03104
03105
03106
                                       typename T::template inject_constant_t<n-1>,
03107
                                       typename stirling_1_helper<T, n-1, k>::type
03108
03109
03110
          } // namespace internal
03111
03116
          template<typename T, int n, int k>
          using stirling_1_signed_t = typename internal::stirling_1_helper<T, n, k>::type;
03117
03118
03123
          template<typename T, int n, int k>
03124
          using stirling_1_unsigned_t = abs_t<typename internal::stirling_1_helper<T, n, k>::type>;
03125
03130
          template<typename T, int n, int k>
03131
          static constexpr typename T::inner_type stirling_1_unsigned_v = stirling_1_unsigned_t<T, n, k>::v;
03132
03137
          template<typename T, int n, int k>
03138
          static constexpr typename T::inner_type stirling_1_signed_v = stirling_1_signed_t<T, n, k>::v;
03139
03140
          namespace internal {
03141
              template<typename T, int n, int k, typename E = void>
03142
              struct stirling_2_helper {};
03143
03144
              template<typename T, int n>
              struct stirling_2_helperTT, n, n, std::enable_if_t<(n >= 0)» {
    using type = typename T::one;
03145
03146
```

```
03147
               };
03148
03149
               template<typename T, int n>
               struct stirling_2_helper<T, n, 0, std::enable_if_t<(n > 0)» {
03150
03151
                   using type = typename T::zero;
03152
               };
03153
03154
               template<typename T, int n>
03155
               struct stirling_2_helper<T, 0, n, std::enable_if_t<(n > 0)» {
03156
                   using type = typename T::zero;
03157
               };
03158
03159
               template<typename T, int n, int k>
03160
               struct stirling_2_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0) && (k < n)» {
                   using type = typename T::template add_t<
03161
                                     typename stirling_2_helper<T, n-1, k-1>::type,
typename T::template mul_t<</pre>
03162
03163
03164
                                         typename T::template inject_constant_t<k>,
03165
                                         typename stirling_2_helper<T, n-1, k>::type
03166
03167
03168
           } // namespace internal
03169
03174
           template<typename T, int n, int k>
03175
           using stirling_2_t = typename internal::stirling_2_helper<T, n, k>::type;
03176
03181
           template<typename T, int n, int k>
03182
          static constexpr typename T::inner_type stirling_2_v = stirling_2_t<T, n, k>::v;
03183
03184
          namespace internal {
03185
               template<tvpename T>
03186
               struct pow_scalar {
03187
                   template<size_t p>
03188
                    \texttt{static constexpr DEVICE INLINED T func(const T\& x) \{ \texttt{return p == 0 ? static\_cast<T>(1) : } \\ 
0.3189
                       p % 2 == 0 ? func < p/2 > (x) * func < p/2 > (x) :
                        x * func<p/2>(x) * func<p/2>(x);
03190
03191
                   }
03192
               };
03193
03194
               template<typename T, typename p, size_t n, typename E = void>
03195
               requires IsEuclideanDomain<T>
0.3196
               struct pow;
03197
03198
               template<typename T, typename p, size_t n>
               struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)» {
03199
03200
                   using type = typename T::template mul_t<
03201
                        typename pow<T, p, n/2>::type,
03202
                        typename pow<T, p, n/2>::type
03203
                   >;
03204
               };
03205
03206
               template<typename T, typename p, size_t n>
03207
               struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)» {
03208
                   using type = typename T::template mul_t<</pre>
03209
03210
                        typename T::template mul t<
                            typename pow<T, p, n/2>::type, typename pow<T, p, n/2>::type
03211
03212
03213
03214
                   >;
03215
               }:
03216
03217
               template<typename T, typename p, size_t n>
03218
               struct pow<T, p, n, std::enable_if_t<n == 0» { using type = typename T::one; };</pre>
03219
             // namespace internal
03220
03225
          template<typename T, typename p, size_t n>
03226
          using pow_t = typename internal::pow<T, p, n>::type;
03227
03232
           template<typename T, typename p, size_t n>
03233
           static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
03234
          template<typename T, size_t p>
static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
03235
03236
      internal::pow scalar<T>::template func(x); }
03237
03238
           namespace internal {
03239
              template<typename, template<typename, size_t> typename, class>
03240
               struct make_taylor_impl;
03241
               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>
03242
03243
                 using type = typename polynomial<FractionField<T>::template val<typename coeff_at<T,
      Is>::type...>;
03245
              } ;
03246
           }
03247
```

```
template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
          using taylor = typename internal::make_taylor_impl<</pre>
03253
03254
03255
              coeff at,
03256
              internal::make index sequence reverse<deg + 1>>::type;
03257
03258
          namespace internal {
03259
              template<typename T, size_t i>
03260
              struct exp_coeff {
03261
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03262
03263
03264
              template<typename T, size_t i, typename E = void>
              struct sin_coeff_helper {};
03265
03266
              template<typename T, size_t i>
03267
              struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
03268
03269
03270
03271
03272
              template<typename T, size_t i>
03273
              using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03274
03275
03276
03277
              template<typename T, size_t i>
03278
              struct sin_coeff {
03279
                  using type = typename sin_coeff_helper<T, i>::type;
03280
03281
              template<typename T, size_t i, typename E = void>
03282
03283
              struct sh coeff helper {};
03284
03285
              template<typename T, size_t i>
03286
              struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0  {
03287
                  using type = typename FractionField<T>::zero;
03288
              };
03289
03290
              template<typename T, size_t i>
03291
              struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03292
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03293
              }:
03294
03295
              template<typename T, size_t i>
03296
              struct sh_coeff {
03297
                  using type = typename sh_coeff_helper<T, i>::type;
03298
03299
03300
              template<typename T, size_t i, typename E = void>
03301
              struct cos_coeff_helper {};
03302
03303
              template<typename T, size_t i>
03304
              struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1\times {
03305
                 using type = typename FractionField<T>::zero;
03306
03307
03308
              template<typename T, size_t i>
03309
              struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03310
                  using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03311
03312
03313
              template<typename T, size_t i>
03314
              struct cos_coeff {
03315
                 using type = typename cos_coeff_helper<T, i>::type;
03316
              };
03317
              template<typename T, size_t i, typename E = void>
03318
              struct cosh_coeff_helper {};
03319
03320
03321
              template<typename T, size_t i>
03322
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03323
                  using type = typename FractionField<T>::zero;
03324
              };
03325
03326
              template<typename T, size_t i>
03327
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03328
                 using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03329
03330
03331
              template<typename T, size_t i>
03332
              struct cosh coeff {
03333
                  using type = typename cosh_coeff_helper<T, i>::type;
03334
03335
03336
              template<typename T, size_t i>
              struct geom_coeff { using type = typename FractionField<T>::one; };
03337
03338
```

```
03340
                template<typename T, size_t i, typename E = void>
03341
                struct atan_coeff_helper;
03342
03343
                template<typename T, size_t i>
struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03344
03345
                    using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>;;
03346
03347
                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;
03348
03349
03350
03351
03352
03353
                template<typename T, size_t i>
03354
                struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
03355
03356
                template<typename T, size_t i, typename E = void>
                struct asin_coeff_helper;
03357
03358
03359
                template<typename T, size_t i>
                struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03360
03361
                    using type = makefraction_t<T,
                         factorial_t<T, i - 1>,
03362
03363
                         typename T::template mul_t<
03364
                             typename T::template val<i>,
03365
                             T::template mul_t<
                                 pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
pow<T, factorial_t<T, i / 2>, 2
03366
03367
03368
03369
03370
                         »;
03371
03372
03373
                template<typename T, size_t i>
                struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
03374
03375
03376
03377
03378
                template<typename T, size_t i>
03379
                struct asin_coeff {
03380
                    using type = typename asin_coeff_helper<T, i>::type;
03381
03382
03383
                template<typename T, size_t i>
03384
                struct lnp1_coeff {
03385
                    using type = makefraction_t<T,
03386
                         alternate_t<T, i + 1>,
03387
                        typename T::template val<i>;;
03388
                };
03389
03390
                template<typename T>
03391
                struct lnpl_coeff<T, 0> { using type = typename FractionField<T>::zero; };
03392
                template<typename T, size_t i, typename E = void>
03393
03394
                struct asinh coeff helper;
03395
03396
                template<typename T, size_t i>
03397
                struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03398
                    using type = makefraction_t<T,</pre>
                         typename T::template mul_t<
03399
                             alternate_t<T, i / 2>,
factorial_t<T, i - 1>
03400
03401
03402
03403
                         typename T::template mul_t<
03404
                             typename T::template mul_t<</pre>
03405
                                  typename T::template val<i>,
pow_t<T, factorial_t<T, i / 2>, 2>
03406
03407
03408
                             pow_t<T, typename T::template inject_constant_t<4>, i / 2>
03409
03410
                    >;
03411
               };
03412
03413
                template<typename T, size t i>
03414
                struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
                   using type = typename FractionField<T>::zero;
03415
03416
03417
03418
                template<typename T, size_t i>
                struct asinh_coeff {
03419
03420
                    using type = typename asinh_coeff_helper<T, i>::type;
03421
03422
03423
                template<typename T, size_t i, typename E = void>
03424
                struct atanh_coeff_helper;
03425
```

```
template<typename T, size_t i>
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03427
03428
                   // 1/i
03429
                   using type = typename FractionField<T>:: template val<
03430
                      typename T::one,
                       typename T::template inject_constant_t<i>;;
03431
03432
              };
03433
03434
              template<typename T, size_t i>
03435
              struct atanh coeff helper<T, i, std::enable if t<(i & 1) == 0» {
03436
                  using type = typename FractionField<T>::zero;
03437
03438
03439
              template<typename T, size_t i>
03440
              struct atanh_coeff {
03441
                  using type = typename atanh_coeff_helper<T, i>::type;
03442
              };
03443
03444
              template<typename T, size_t i, typename E = void>
03445
              struct tan_coeff_helper;
03446
03447
              template<typename T, size_t i>
              struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
    using type = typename FractionField<T>::zero;
03448
03449
03450
03451
              template<typename T, size_t i>
03452
03453
              struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
03454
              private:
                   // 4^((i+1)/2)
03455
                   using _4p = typename FractionField<T>::template inject_t<</pre>
03456
03457
                       pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
03458
                   // 4^((i+1)/2) - 1
03459
                   using _4pm1 = typename FractionField<T>::template
     sub_t<_4p, typename FractionField<T>::one>;
    // (-1)^((i-1)/2)
03460
03461
                   using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
                   using dividend = typename FractionField<T>::template mul_t<
03462
03463
                       altp,
03464
                       FractionField<T>::template mul_t<</pre>
03465
                       _4p,
                       FractionField<T>::template mul t<
03466
03467
                       4pm1,
03468
                       bernoulli_t<T, (i + 1)>
03469
03470
03471
03472
              public:
                  using type = typename FractionField<T>::template div t<dividend.
03473
03474
                       typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03475
              };
03476
03477
              template<typename T, size_t i>
03478
              struct tan_coeff {
03479
                  using type = typename tan_coeff_helper<T, i>::type;
03480
03481
03482
              template<typename T, size_t i, typename E = void>
03483
              struct tanh_coeff_helper;
03484
03485
              template<typename T, size_t i>
              struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
03486
03487
                  using type = typename FractionField<T>::zero;
03488
03489
03490
              template<typename T, size_t i>
03491
              struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
03492
              private:
03493
                  using _4p = typename FractionField<T>::template inject_t<</pre>
                       pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
03494
03495
                  using _4pm1 = typename FractionField<T>::template
      sub_t<_4p, typename FractionField<T>::one>;
03496
                  using dividend =
03497
                       typename FractionField<T>::template mul t<</pre>
03498
                           _4p,
                           typename FractionField<T>::template mul_t<</pre>
03499
03500
                               _4pm1,
03501
                               bernoulli_t<T, (i + 1) >>::type;
              public:
03502
                  using type = typename FractionField<T>::template div_t<dividend,</pre>
03503
                      FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03504
03505
              };
03506
03507
              template<typename T, size_t i>
03508
              struct tanh_coeff {
                   using type = typename tanh_coeff_helper<T, i>::type;
03509
03510
              };
```

```
} // namespace internal
03512
03516
                template<typename Integers, size_t deg>
03517
                using exp = taylor<Integers, internal::exp_coeff, deg>;
03518
03522
               template<typename Integers, size_t deg>
using expm1 = typename polynomial<FractionField<Integers>>::template sub_t
03523
03524
                       exp<Integers, deg>,
03525
                      typename polynomial<FractionField<Integers>>::one>;
03526
03530
                template<typename Integers, size_t deg>
03531
                using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
03532
03536
                template<typename Integers, size_t deg>
03537
                using atan = taylor<Integers, internal::atan_coeff, deg>;
03538
03542
                template<typename Integers, size_t deg>
03543
                using sin = taylor<Integers, internal::sin_coeff, deq>;
03544
03548
                template<typename Integers, size_t deg>
03549
                using sinh = taylor<Integers, internal::sh_coeff, deg>;
03550
03555
                template<typename Integers, size_t deg>
03556
                using cosh = taylor<Integers, internal::cosh_coeff, deg>;
03557
03562
                template<typename Integers, size_t deg>
03563
                using cos = taylor<Integers, internal::cos_coeff, deg>;
03564
03569
                template<typename Integers, size_t deg>
03570
                using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
03571
03576
                template<typename Integers, size t deg>
03577
                using asin = taylor<Integers, internal::asin_coeff, deg>;
03578
03583
                template<typename Integers, size_t deg>
03584
                using asinh = taylor<Integers, internal::asinh_coeff, deg>;
03585
03590
                template<typename Integers, size_t deg>
03591
                using atanh = taylor<Integers, internal::atanh_coeff, deg>;
03592
03597
                template<typename Integers, size_t deg>
03598
                using tan = taylor<Integers, internal::tan_coeff, deg>;
03599
03604
                template<typename Integers, size_t deg>
                using tanh = taylor<Integers, internal::tanh_coeff, deg>;
03605
03606 } // namespace aerobus
03607
03608 // continued fractions
03609 namespace aerobus {
               template<int64_t... values>
03612
03613
                struct ContinuedFraction {};
03614
03617
                template<int64_t a0>
03618
                struct ContinuedFraction<a0> {
                      using type = typename q64::template inject_constant_t<a0>;
03620
                      static constexpr double val = static_cast<double>(a0);
03622
03623
03624
03628
                template<int64_t a0, int64_t... rest>
03629
                struct ContinuedFraction<a0, rest...> {
03631
                      using type = g64::template add t<
03632
                                   typename q64::template inject_constant_t<a0>,
03633
                                   typename q64::template div_t<
03634
                                          typename q64::one,
03635
                                          typename ContinuedFraction<rest...>::type
03636
03637
03639
                      static constexpr double val = type::template get<double>();
03640
               };
03641
03645
                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>;
03647
               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>;
03649
               using SORT2 fraction =
         using SQRT3_fraction
03651
         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
03652 } // namespace aerobus
03653
03654 // known polynomials
03655 namespace aerobus {
03656
                // CChebyshev
03657
                namespace internal {
                      template<int kind, size_t deg, typename I>
03658
03659
                      struct chebyshev helper {
```

```
using type = typename polynomial<I>::template sub_t<
03661
                       typename polynomial<I>::template mul_t<</pre>
03662
                           typename polynomial<I>::template mul_t<</pre>
03663
                               typename polynomial<I>::template inject_constant_t<2>,
03664
                               typename polynomial<I>::X>
                           typename chebyshev_helper<kind, deg - 1, I>::type
03665
03666
03667
                       typename chebyshev_helper<kind, deg - 2, I>::type
03668
03669
              } ;
03670
03671
              template<tvpename I>
03672
              struct chebyshev_helper<1, 0, I> {
03673
                  using type = typename polynomial<I>::one;
03674
03675
03676
              template<typename I>
              struct chebyshev_helper<1, 1, I> {
    using type = typename polynomial<I>::X;
03677
03678
03679
              };
03680
03681
              template<typename I>
              struct chebyshev_helper<2, 0, I> {
03682
03683
                  using type = typename polynomial<I>::one;
03684
03685
03686
              template<typename I>
03687
              struct chebyshev_helper<2, 1, I> {
03688
                  using type = typename polynomial<I>::template mul_t<</pre>
                       typename polynomial<I>::template inject_constant_t<2>,
03689
03690
                       typename polynomial<I>::X>;
03691
              };
03692
          } // namespace internal
03693
03694
          // Laguerre
03695
          namespace internal {
03696
              template<size_t deg, typename I>
              struct laguerre_helper {
03697
03698
                  using Q = FractionField<I>;
03699
                   using PQ = polynomial<Q>;
03700
03701
               private:
                  // Lk = (1 / k) * ((2 * k - 1 - x) * 1km1 - (k - 2) Lkm2)
03702
                  using lnm2 = typename laguerre_helper<deg - 2, I>::type;
03703
03704
                  using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03705
                   // -x + 2k-1
03706
                   using p = typename PQ::template val<
03707
                       typename Q::template inject_constant_t<-1>,
03708
                       typename Q::template inject_constant_t<2 * deg - 1»;</pre>
                   // 1/n
03709
03710
                  using factor = typename PQ::template inject_ring_t<</pre>
                       typename Q::template val<typename I::one, typename I::template
03711
      inject_constant_t<deg>>;
03712
03713
               public:
03714
                  using type = typename PQ::template mul_t <</pre>
03715
                       factor,
03716
                       typename PQ::template sub_t<
03717
                           typename PQ::template mul_t<</pre>
03718
03719
                               1 nm1
03720
03721
                           typename PQ::template mul_t<
03722
                               typename PQ::template inject_constant_t<deg-1>,
03723
03724
03725
03726
                  >;
03727
              };
03728
03729
              template<typename I>
03730
              struct laguerre_helper<0, I> {
03731
                  using type = typename polynomial<FractionField<I>::one;
03732
03733
03734
              template<typename I>
03735
              struct laguerre_helper<1, I> {
03736
              private:
03737
                  using PQ = polynomial<FractionField<I>;
03738
               public:
03739
                  using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
03740
03741
          } // namespace internal
03742
03743
          // Bernstein
03744
          namespace internal {
03745
              template<size t i, size t m, typename I, typename E = void>
```

```
03746
              struct bernstein_helper {};
03747
03748
              template<typename I>
              struct bernstein_helper<0, 0, I> {
03749
03750
                  using type = typename polynomial<I>::one;
03751
              };
03752
03753
              template<size_t i, size_t m, typename I>
              03754
03755
03756
               private:
03757
                 using P = polynomial<I>;
03758
               public:
03759
                 using type = typename P::template mul_t<
03760
                          typename P::template sub_t<typename P::one, typename P::X>,
03761
                          typename bernstein_helper<i, m-1, I>::type>;
03762
              };
03763
03764
              template<size_t i, size_t m, typename I>
03765
              struct bernstein_helper<i, m, I, std::enable_if_t<
03766
                          (m > 0) \&\& (i == m) \gg {
               private:
03767
                 using P = polynomial<I>;
03768
03769
               public:
03770
                  using type = typename P::template mul_t<
03771
                          typename P::X,
03772
                          typename bernstein_helper<i-1, m-1, I>::type>;
03773
03774
03775
              template<size_t i, size_t m, typename I>
              struct bernstein_helper<i, m, I, std::enable_if_t<
(m > 0) && (i > 0) && (i < m)» {
03776
03778
03779
                  using P = polynomial<I>;
03780
               public:
03781
                 using type = typename P::template add_t<
03782
                          typename P::template mul_t<
03783
                              typename P::template sub_t<typename P::one, typename P::X>,
03784
                              typename bernstein_helper<i, m-1, I>::type>,
03785
                          typename P::template mul_t<
03786
                              typename P::X,
                              typename bernstein_helper<i-1, m-1, I>::type»;
03787
03788
          } // namespace internal
03789
03790
03791
          // AllOne polynomials
03792
          namespace internal {
03793
              template<size_t deg, typename I>
              struct AllOneHelper {
03794
03795
                  using type = aerobus::add_t<
03796
                      typename polynomial<I>::one,
03797
                      typename aerobus::mul_t<</pre>
03798
                          typename polynomial<I>::X,
03799
                          typename AllOneHelper<deg-1, I>::type
03800
03801
              };
03803
              template<typename I>
03804
              struct AllOneHelper<0, I> {
03805
                  using type = typename polynomial<I>::one;
03806
03807
          } // namespace internal
03808
03809
          // Bessel polynomials
03810
          namespace internal {
03811
              template<size_t deg, typename I>
03812
              struct BesselHelper {
03813
               private:
                  using P = polynomial<I>;
03814
                  using factor = typename P::template monomial_t<
03815
03816
                    typename I::template inject_constant_t<(2*deg - 1)>,
                      1>;
03817
               public:
03818
                  using type = typename P::template add_t<
03819
03820
                      typename P::template mul_t<</pre>
03821
                          factor.
03822
                          typename BesselHelper<deg-1, I>::type
03823
03824
                      typename BesselHelper<deg-2, I>::type
03825
                  >:
03826
              };
03827
              template<typename I>
03828
03829
              struct BesselHelper<0, I> {
03830
                  using type = typename polynomial<I>::one;
03831
              };
03832
```

```
template<typename I>
                         struct BesselHelper<1, I> {
03834
                          private:
03835
03836
                               using P = polynomial<I>;
03837
                           public:
03838
                               using type = typename P::template add t<
03839
                                       typename P::one,
03840
                                       typename P::X
03841
03842
                 } // namespace internal
03843
03844
03845
                 namespace known polynomials {
                         enum hermite_kind {
03847
03849
                               probabilist,
03851
                               physicist
03852
                        };
03853
                 }
03854
03855
                  // hermite
                  namespace internal {
03856
03857
                         template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03858
                         struct hermite_helper {};
03859
03860
                         template<size_t deg, typename I>
                         struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
03861
03862
03863
                               using hnm1 = typename hermite_helper<deg - 1,</pre>
          known_polynomials::hermite_kind::probabilist, I>::type;
03864
                               using hnm2 = typename hermite_helper<deg - 2,
          known_polynomials::hermite_kind::probabilist, I>::type;
03865
03866
                           public:
03867
                               using type = typename polynomial<I>::template sub_t<</pre>
03868
                                       \label{typename} \verb|polynomial<I>::template mul\_t<typename polynomial<I>::X, \verb|hnm1>|, typename polynomial<I>::X, typename polynomial<II>::X, typename polynomial<II>::X, typename polynomial<II:X, typename polynomial<II
                                       typename polynomial<I>::template mul_t<
03869
03870
                                               typename polynomial<I>::template inject_constant_t<deg - 1>,
03871
                                              hnm2
03872
03873
                               >;
03874
                         };
03875
03876
                         template<size t deg, typename I>
03877
                         struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03878
                          private:
03879
                                using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
          I>::type;
                                using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
03880
          I>::tvpe;
03881
03882
                           public:
03883
                                using type = typename polynomial<I>::template sub_t<</pre>
03884
                                        // 2X Hn-1
03885
                                       typename polynomial<I>::template mul_t<
03886
                                               typename pi64::val<typename I::template inject_constant_t<2>,
03887
                                              typename I::zero>, hnm1>,
03888
03889
                                       typename polynomial<I>::template mul_t<
03890
                                               typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03891
                                              hnm2
03892
                                       >
03893
                               >;
03894
                         };
03895
03896
                         template<typename I>
03897
                         struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03898
                               using type = typename polynomial<I>::one;
03899
                         };
03900
03901
                         template<typename I>
03902
                         struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03903
                                using type = typename polynomial<I>::X;
03904
                         };
03905
03906
                         template<typename I>
03907
                         struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03908
                               using type = typename pi64::one;
03909
03910
03911
                         template<tvpename T>
03912
                         struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03913
                                     2X
03914
                                using type = typename polynomial<I>::template val<
03915
                                       typename I::template inject_constant_t<2>,
03916
                                       typename I::zero>;
03917
                       // namespace internal
03918
```

```
03919
03920
03921
          namespace internal {
03922
               template<size_t n, typename I>
03923
               struct legendre_helper {
03924
                private:
                   using Q = FractionField<I>;
03925
03926
                   using PQ = polynomial<Q>;
03927
                   // 1/n constant
                   // (2n-1)/n X
03928
                   using fact_left = typename PQ::template monomial_t<</pre>
03929
                       makefraction_t<I,
typename I::template inject_constant_t<2*n-1>,
03930
03931
03932
                            typename I::template inject_constant_t<n>
03933
                       >,
                   1>;
03934
                   // (n-1) / n
03935
                   using fact_right = typename PQ::template val<</pre>
03936
                       makefraction_t<I,
03937
03938
                            typename I::template inject_constant_t<n-1>,
03939
                            typename I::template inject_constant_t<n>>;
03940
03941
                public:
                   using type = PQ::template sub_t<
03942
03943
                            typename PQ::template mul_t<
03944
                               fact_left,
03945
                                typename legendre_helper<n-1, I>::type
03946
03947
                            typename PQ::template mul_t<
03948
                                fact_right,
03949
                                typename legendre helper<n-2, I>::type
03950
03951
                       >;
03952
               };
03953
03954
               template<typename I>
               struct legendre_helper<0, I> {
    using type = typename polynomial<FractionField<I»::one;</pre>
03955
03956
03957
03958
03959
               template<typename I>
               struct legendre_helper<1, I> {
03960
                  using type = typename polynomial<FractionField<I>::X;
03961
03962
03963
           } // namespace internal
03964
03965
           // bernoulli polynomials
03966
          namespace internal {
              template<size_t n>
struct bernoulli_coeff {
03967
03968
03969
                   template<typename T, size_t i>
03970
                   struct inner {
03971
                    private:
03972
                       using F = FractionField<T>;
03973
                    public:
03974
                       using type = typename F::template mul t<
03975
                            typename F::template inject_ring_t<combination_t<T, i, n»,
03976
                            bernoulli_t<T, n-i>
03977
03978
                   };
03979
               };
03980
          } // namespace internal
03981
03982
          namespace internal {
03983
               template<size_t n>
03984
               struct touchard_coeff {
03985
                   template<typename T, size_t i>
03986
                   struct inner {
03987
                       using type = stirling_2_t<T, n, i>;
03988
                   };
03989
03990
          } // namespace internal
03991
03992
          namespace internal {
03993
               template<typename I = aerobus::i64>
03994
               struct AbelHelper {
03995
               private:
03996
                   using P = aerobus::polynomial<I>;
03997
03998
                public:
03999
                  // to keep recursion working, we need to operate on a*n and not just a
04000
                   template<size_t deg, I::inner_type an>
04001
                   struct Inner {
04002
                       // abel(n, a) = (x-an) * abel(n-1, a)
04003
                       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>>
04004
04005
```

```
>;
04007
04008
                            // abel(0, a) = 1
04009
04010
                            template<I::inner_type an>
                            struct Inner<0, an> {
   using type = P::one;
04011
04012
04013
04014
                            // abel(1, a) = X
04015
04016
                            template<I::inner_type an>
04017
                            struct Inner<1, an> {
    using type = P::X;
04018
04019
04020
                      };
04021
             } // namespace internal
04022
04024
               namespace known polynomials {
04034
                      template<size_t n, auto a, typename I = aerobus::i64>
04035
                      using abel = typename internal::AbelHelper<I>::template Inner<n, a*n>::type;
04036
04044
                      template <size_t deg, typename I = aerobus::i64>
04045
                      using chebyshev T = typename internal::chebyshev helper<1, deg, I>::type;
04046
04056
                      template <size_t deg, typename I = aerobus::i64>
04057
                      using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
04058
04068
                      template <size_t deg, typename I = aerobus::i64>
04069
                      using laquerre = typename internal::laquerre_helper<deg, I>::type;
04070
04077
                      template <size_t deg, typename I = aerobus::i64>
                      using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
04078
        I>::type;
04079
04086
                      template <size_t deg, typename I = aerobus::i64>
04087
                      using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
04099
                      template<size_t i, size_t m, typename I = aerobus::i64>
04100
                      using bernstein = typename internal::bernstein_helper<i, m, I>::type;
04101
04111
                      template<size_t deg, typename I = aerobus::i64>
                      using legendre = typename internal::legendre_helper<deg, I>::type;
04112
04113
04123
                      template<size_t deg, typename I = aerobus::i64>
04124
                      using bernoulli = taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
04125
                      template<size_t deg, typename I = aerobus::i64>
04132
                      using allone = typename internal::AllOneHelper<deg, I>::type;
04133
04134
04142
                      template<size_t deg, typename I = aerobus::i64>
04143
                      using bessel = typename internal::BesselHelper<deg, I>::type;
04144
04152
                      template<size_t deg, typename I = aerobus::i64>
                      using touchard = taylor<I, internal::touchard_coeff<deq>::template inner, deq>;
04153
                    // namespace known_polynomials
04154
04155 } // namespace aerobus
04156
04157
04158 #ifdef AEROBUS CONWAY IMPORTS
04159
04160 // conway polynomials
04161 namespace aerobus {
         template<int p, int n>
04165
04166
               struct ConwayPolynomial {};
04167
04168 #ifndef DO NOT DOCUMENT
            #define ZPZV ZPZ::template val
04169
               #define POLYV aerobus::polynomial<ZPZ>::template val
04170
04171
                template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
        POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
04172
               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 =
04173
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1), ; // NOLINT template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
04174
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
04175
               template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT
04176
               template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<2>; yPZV<2>, ZPZV<2>; yPZV<2>; yPZV<2>; yPZV<2>; yPZV<2>; yPZV<3>; yPZV<3
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
04178
               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
04179
         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<0>, ZPZV<1»; }; //</pre>
```

```
NOLINT
                                            template<> struct ConwayPolynomial<2, 10> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , Z
                                             ZPZV<1»; }; // NOLINT</pre>
                                                                            template<> struct ConwayPolynomial<2, 11> { using ZPZ = aerobus::zpz<2>; using type
                                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                                                                            template<> struct ConwayPolynomial<2, 12> { using ZPZ = aerobus::zpz<2>; using type
                                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                            template<> struct ConwayPolynomial<2, 13> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0>, ZPZV<0 , Z
04183
                                             ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                                                           template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type =
                                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                                             ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT
template<> struct ConwayPolynomial<2, 15> { using ZPZ = aerobus::zpz<2>; using type =
                                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                                                                            template<> struct ConwayPolynomial<2, 16> { using ZPZ = aerobus::zpz<2>; using type
                                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                             04187
                                                                        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
                                                                            template<> struct ConwayPolynomial<2, 18> { using ZPZ = aerobus::zpz<2>; using type =
                                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>,
                                             template<> struct ConwayPolynomial<2, 19> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
                                            NOLINT
                                                                           template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type
                                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                               ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1</pre>; };
                                             // NOLINT
04191
                                                                           template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
                                            POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
 04192
                                                                           template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
                                             POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2»; }; // NOLINT
 04193
                                                                       template<> struct ConwayPolynomial<3, 3> { using ZPZ = aerobus::zpz<3>; using type =
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT
  template<> struct ConwayPolynomial<3, 4> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; }; // NOLINT</pre>
 04194
                                                                            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
 04196
                                                                       template<> struct ConwayPolynomial<3, 6> { using ZPZ = aerobus::zpz<3>; using type =
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2»; }; // NOLINT
 04197
                                                                           template<> struct ConwayPolynomial<3, 7> { using ZPZ = aerobus::zpz<3>; using type =
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<1»; }; // NOLINT
04198
                                                                          template<> struct ConwayPolynomial<3, 8> { using ZPZ = aerobus::zpz<3>; using type =
                                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2
 04199
                                                                       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
04200
                                            template<> struct ConwayPolynomial<3, 10> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
                                             ZPZV<2»: }: // NOLINT
                                           template<>> struct ConwayPolynomial<3, 11> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV
                                                                      template<> struct ConwayPolynomial<3, 13> { using ZPZ = aerobus::zpz<3>; using type =
                                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                             ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<3, 14> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3 , Z
04204
                                             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<0>, ZPZV<0>, 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 =
04206
                                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                             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<2>, ZPZV<1»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<3, 18> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
04208
                                             ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>; };
                                            template<> struct ConwayPolynomial<3, 19> { using ZPZ = aerobus::zpz<3>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                              \texttt{ZPZV} < \texttt{0} >, \ \texttt{ZPZV} < \texttt{2} >, \ \texttt{ZPZV} < \texttt{2} >, \ \texttt{ZPZV} < \texttt{3} >, \ \texttt{2} >, \ \texttt{2
                                            NOLINT
                                                                        template<> struct ConwayPolynomial<3, 20> { using ZPZ = aerobus::zpz<3>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
                               // NOLINT
04211
                                                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 =
04212
                             POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<5, 3> { using ZPZ = aerobus::zpz<5>; using type =
04213
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
04214
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<5, 5> { using ZPZ = aerobus::zpz<5>; using type =
04215
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<3»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<5, 6> { using ZPZ = aerobus::zpz<5>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<5, 7> { using ZPZ = aerobus::zpz<5>; using type =
04217
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
04218
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<2; };
04219
                                                  template<> struct ConwayPolynomial<5, 9> { using ZPZ = aerobus::zpz<5>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<3»; }; //
                              NOLINT
04220
                                                template<> struct ConwayPolynomial<5, 10> { using ZPZ = aerobus::zpz<5>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<1>,
                              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</pre>
04222
                                                   template<> struct ConwayPolynomial<5, 12> { using ZPZ = aerobus::zpz<5>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<2»; }; // NOLINT
04223
                                                   template<> struct ConwayPolynomial<5, 13> { using ZPZ = aerobus::zpz<5>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<5, 14> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>,
04224
                             ZPZV<2>, ZPZV<3>, ZPZV<0>, ZPZV<1>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<5, 15> { using ZPZ = aerobus::zpz<5>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<3»; }; // NOLINT</pre>
04226
                                                template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type =
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4 , ZPZV<4
                                                  template<> struct ConwayPolynomial<5, 17> { using ZPZ = aerobus::zpz<5>; using type =
04227
                               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>
04228
                                                template<> struct ConwayPolynomial<5, 18> { using ZPZ = aerobus::zpz<5>; using type =
                              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
                              ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<2», }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<5, 19> { using ZPZ = aerobus::zpz<5; using type = POLYV<ZPZV<1>, ZPZV<0>, ZP
04229
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<3»; }; //</pre>
                             template<> struct ConwayPolynomial<5, 20> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3</pre>
04230
                               ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<0>, ZPZV<4>, ZPZV<1>, ZPZV<2>; };
                                                 template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
                              POLYV<ZPZV<1>, ZPZV<4»; }; // NOLINT
04232
                                                   template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
                             POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<7, 3> { using ZPZ = aerobus::zpz<7>; using type =
04233
                             POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4»; }; // NOLINT template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
04235
                             template<> struct ConwayPolynomial<7, 6> { using ZPZ = aerobus::zpz<7>; using type =
04236
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<4>, ZPZV<6>, ZPZV<6>, ZPZV<3»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<7, 7> { using ZPZ = aerobus::zpz<7>; using type
04237
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<4»; }; // NOLINT
04238
                                               template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<6>, ZPZV<2>, ZPZV<2»; }; // NOLINT
04239
                                                template<> struct ConwayPolynomial<7, 9> { using ZPZ = aerobus::zpz<7>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
04240
                                                template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<3>,
                              ZPZV<3»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<7, 11> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
04241
                              ZPZV<1>, ZPZV<4»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<7, 12> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<5>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<4>, ZPZV<0>,
                              ZPZV<5>, ZPZV<0>, ZPZV<3>; // NOLINT
                             template<> struct ConwayPolynomial<7, 13> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
```

```
ZPZV<0>, ZPZV<6>, ZPZV<0>, ZPZV<4»; };</pre>
                                                                                                                                                                                                   // NOLINT
                         template<> struct ConwayPolynomial<7, 14> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                         template<> struct ConwayPolynomial<7, 155 { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>,
                          ZPZV<6>, ZPZV<6>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<4»; }; // NOLINT
                                           template<> struct ConwayPolynomial<7, 16> { using ZPZ = aerobus::zpz<7>; using type
                        Template() Struct ConwayFolynomial(), 100 { using ZFZ = derouds.:2pz*//, using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5>, ZPZV<5>, ZPZV<3>, ZPZV<4>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<4>, ZPZV<3»; }; // NOLINT template() struct ConwayPolynomial(7, 17> { using ZPZ = derobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV
04247
                          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<1>,
                         ZPZV<6>, ZPZV<5>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<2>, ZPZV<2>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<7, 19> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<0>, ZPZV<4»; }; //</pre>
                         template<> struct ConwayPolynomial<7, 20> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
                          ZPZV<2>, ZPZV<5>, ZPZV<2>, ZPZV<3>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<1>, ZPZV<3>; };
                          // NOLINT
04251
                                           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
04253
                                          template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<9»; }; // NOLINT template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type =
04254
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2»; };
                                                                                                                                                                                                                                                                       // NOLINT
                                          template<> struct ConwayPolynomial<11, 5> { using ZPZ = aerobus::zpz<11>; using type =
04255
                         04256
                                          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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<9»; }; // NOLINT
04257
04258
                                          template<> struct ConwayPolynomial<11, 8> { using ZPZ = aerobus::zpz<11>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<1>, ZPZV<7>, ZPZV<2»; };
04259
                                        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
                          NOLINT
04260
                                         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>
04261
                                         template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type
04262
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<2»; }; // NOLINT
04263
                                        template<> struct ConwayPolynomial<11, 13> { using ZPZ = aerobus::zpz<11>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         template<> struct ConwayPolynomial<11, 14> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<6>, ZPZV<6 , ZPZ
04264
                          ZPZV<4>, ZPZV<8>, ZPZV<6>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<11, 15> { using ZPZ = aerobus::zpz<11>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
                          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
                          template<> struct ConwayPolynomial<11, 18> { using ZPZ = aerobus::zpz<11>; using type =
04268
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<5>, ZPZV<5>, ZPZV<5
                          ZPZV<3>, ZPZV<9>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<9>, ZPZV<8>, ZPZV<2>, ZPZV<2»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<11, 19> { using ZPZ = aerobus::zpz<11>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<9»; }; //</pre>
                          NOLINT
                         template<> struct ConwayPolynomial<11, 20> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
                          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
                                          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 =
04272
                        POLYV<ZPZV<1>, ZPZV<12>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<13, 3> { using ZPZ = aerobus::zpz<13>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11»; };
                                                                                                                                                                                                                                  // NOLINT
04274
                                       template<> struct ConwayPolynomial<13, 4> { using ZPZ = aerobus::zpz<13>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<13, 5> { using ZPZ = aerobus::zpz<13>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<11»; };
                                               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 =
04277
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<11»; }; // NOLINT
04278
                                             template<> struct ConwayPolynomial<13, 8> { using ZPZ = aerobus::zpz<13>; using type =
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<1>, ZPZV<2>, ZPZV<3>, ZPZV<3>; };
04279
                                               template<> struct ConwayPolynomial<13, 9> { using ZPZ = aerobus::zpz<13>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<8>, ZPZV<12>, ZPZV<12>, ZPZV<11>; };
                            // NOLINT
04280
                                            template<> struct ConwayPolynomial<13, 10> { using ZPZ = aerobus::zpz<13>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<5>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>,
                            ZPZV<2»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<13, 11> { using ZPZ = aerobus::zpz<13>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<3>, ZPZV<11»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<13, 12> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<8>, ZPZV<11>, ZPZV<3>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5>; // NOLINT
                                              template<> struct ConwayPolynomial<13, 13> { using ZPZ = aerobus::zpz<13>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                            template<> struct ConwayPolynomial<13, 14> { using ZPZ = aerobus::zpz<13>; using type =
04284
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<11>, ZPZV<7>, ZPZV<10>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type
                            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<12>, ZPZV<13>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<6>, ZPZV<11»; }; // NOLINT</pre>
04288
                                            template<> struct ConwayPolynomial<13, 18> { using ZPZ = aerobus::zpz<13>; using type =
                           POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                            ZPZV<11>, ZPZV<9>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ; }; // NOLINT
                                              template<> struct ConwayPolynomial<13, 19> { using ZPZ = aerobus::zpz<13>; using type =
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<9>, ZPZV<11»; };</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<4>, ZPZV<8>, ZPZV<5>, ZPZV<1>, ZPZV<5</pre>
; ;;
04290
04291
                                              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 =
04292
                           POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»; }; // NOLINT
                                              template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
04293
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
                           template<> struct ConwayPolynomial<17, 5> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT</pre>
04295
                                              template<> struct ConwayPolynomial<17, 6> { using ZPZ = aerobus::zpz<17>; using type =
04296
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<10>, ZPZV<3>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<17, 7> { using ZPZ = aerobus::zpz<17>; using type
04297
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<14»; }; // NOLINT template<> struct ConwayPolynomial<17, 8> { using ZPZ = aerobus::zpz<17>; using type =
04298
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<3, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<3, ZPZV<3, ZPZV<12, ZPZV<1>, ZPZV<3, ZPZV<3, ZPZV<3, ZPZV<12, ZPZV<1>, ZPZV<3, ZPZV<3, ZPZV<1>, ZPZV<1
04299
04300
                                          template<> struct ConwayPolynomial<17, 10> { using ZPZ = aerobus::zpz<17>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<12>,
                            ZPZV<3»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<17, 11> { using ZPZ = aerobus::zpz<17>; using type =
04301
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<5>, ZPZV<14»; };</pre>
                                                                                                                                     // NOLINT
                                              template<> struct ConwayPolynomial<17, 12> { using ZPZ = aerobus::zpz<17>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<13>, ZPZV<6>,
                            ZPZV<14>, ZPZV<9>, ZPZV<3»; }; // NOLINT</pre>
04303
                                              template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           template<> struct ConwayPolynomial<17, 14> { using ZPZ = aerobus::zpz<17>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
                            ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::zpz<17>; using type :
04305
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>,
                           ZPZV<4>, ZPZV<16>, ZPZV<6>, ZPZV<14>, ZPZV<14>, ZPZV<14»; }; // NOLINT
    template<> struct ConwayPolynomial<17, 16> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
                           ZPZV<5>, ZPZV<2>, ZPZV<12>, ZPZV<13>, ZPZV<12>, ZPZV<1>, ZPZV<3»; }; // NOLINT
    template<> struct ConwayPolynomial<17, 17> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04307
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<14»; };</pre>
                               template<> struct ConwayPolynomial<17, 18> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1 ,
                                template<> struct ConwayPolynomial<17, 19> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0</pre>
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 7P7V<0>.
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<14»; }; //</pre>
                               template<> struct ConwayPolynomial<17, 20> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<15>, ZPZV<16>, ZPZV<14>, ZPZV<13>, ZPZV<3>, ZPZV<14>, ZPZV<15>, ZPZV<15>, ZPZV<15, Z
                                ZPZV<3»: }: // NOLINT
                                                    template<> struct ConwayPolynomial<19, 1> { using ZPZ = aerobus::zpz<19>; using type =
04311
                                POLYV<ZPZV<1>, ZPZV<17»; }; // NOLINT
 04312
                                                  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 =
04313
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17»; };
                                                                                                                                                                                                                                                                                // NOLINT
                                                    template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<19, 5> { using ZPZ = aerobus::zpz<19>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<17»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<19, 6> { using ZPZ = aerobus::zpz<19>; using type =
04316
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<17>, ZPZV<6>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 7> { using ZPZ = aerobus::zpz<19>; using type
 04317
                               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<17»; }; //
 04318
                                                  template<> struct ConwayPolynomial<19, 8> { using ZPZ = aerobus::zpz<19>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<10>, ZPZV<3>, ZPZV<2»; };
                               template<> struct ConwayPolynomial<19, 9> { using ZPZ = aerobus::zpz<19>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<14>, ZPZV<16>, ZPZV<17»; };</pre>
 04319
                                // NOLINT
04320
                                                     template<> struct ConwayPolynomial<19, 10> { using ZPZ = aerobus::zpz<19>; using type
                                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<3>, ZPZV<4>,
                                ZPZV<2»; }; // NOLINT</pre>
                               \label{template} $$ \text{template}$ <> $$ \text{struct ConwayPolynomial}<19, 11> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZP
 04321
                                ZPZV<8>, ZPZV<17»; };</pre>
                                                                                                                                                       // NOLINT
                                                    template<> struct ConwayPolynomial<19, 12> { using ZPZ = aerobus::zpz<19>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3
                                 ZPZV<16>, ZPZV<7>, ZPZV<2»; };</pre>
                                                                                                                                                                                                   // NOLINT
04323
                                                  template<> struct ConwayPolynomial<19, 13> { using ZPZ = aerobus::zpz<19>; using type =
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
04324
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0), ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<11>,
                                 ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<7>, ZPZV<2»; }; // NOLINT</pre>
04325
                                                   template<> struct ConwayPolynomial<19, 15> { using ZPZ = aerobus::zpz<19>; using type =
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>; // NOLINT template<> struct ConwayPolynomial<19, 16> { using ZPZ = aerobus::zpz<19>; using type =
04326
                                POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<13>, ZPZV<0>, ZPZV<15>, ZPZV<9>, ZPZV<6>, ZPZV<14>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<19, 17> { using ZPZ = aerobus::zpz<19>; using type
04327
                               POLYY<ZPZV<0>, ZPZV<0>, ZPZV<1, ZPZV<0>, ZPZV<0 , ZPZV<0 
04328
                                                    template<> struct ConwayPolynomial<19, 18> { using ZPZ = aerobus::zpz<19>; using type
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<16>, ZPZV<5>, ZPZV<16 , ZPZV<5 , ZPZV<16 , ZPZV<5 , ZPZV<16 , ZPZV<5 , ZPZV<5 , ZPZV<16 , ZPZV<5 , 
                                                  template<> struct ConwayPolynomial<19, 19> { using ZPZ = aerobus::zpz<19>; using type =
                               POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<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<11>, ZPZV<2»;</pre>
                                }; // NOLINT
04331
                                                     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 =
04332
                               POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
 04333
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; ); // NOLINT template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
 04334
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT
                              template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<18»; }; // NOLINT
 04335
                                                   template<> struct ConwayPolynomial<23, 6> { using ZPZ = aerobus::zpz<23>; using type =
 04336
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<9>, ZPZV<5>, ZPZV<5»; }; // NOLINT
                             template<> struct ConwayPolynomial<23, 7> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<18»; }; // NOLINT</pre>
 04337
                                                    template<> struct ConwayPolynomial<23, 8> { using ZPZ = aerobus::zpz<23>; using type =
 04338
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<20>, ZPZV<5>, ZPZV<3>, ZPZV<5>; };
                             template<> struct ConwayPolynomial<23, 9> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<9>, ZPZV<18»; };
                                // NOLINT
                               template<> struct ConwayPolynomial<23, 10> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<15>, ZPZV<15>, ZPZV<15>, ZPZV<6>, ZPZV<6>, ZPZV<1>,
 04340
```

```
ZPZV<5»; };</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</pre>
                            template<> struct ConwayPolynomial<23, 12> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<21>, ZPZV<15>, ZPZV<14>, ZPZV<12>,
ZPZV<18>, ZPZV<12>, ZPZV<18>, ZPZV<5»; }; // NOLINT</pre>
04342
                                                template<> struct ConwayPolynomial<23, 13> { using ZPZ = aerobus::zpz<23>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<18»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<23, 14> { using ZPZ = aerobus::zpz<23>; using type =
04344
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<1>, ZPZV<5>, ZPZV<1>,
                            ZPZV<18>, ZPZV<19>, ZPZV<2), ZPZV<22>, ZPZV<5>; }; // NOLINT
template<> struct ConwayPolynomial<23, 15> { using ZPZ = aerobus::zpz<23>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<8>, ZPZV<15>, ZPZV<9>, ZPZV<7>, ZPZV<18>, ZPZV<18»; }; // NOLINT
template<> struct ConwayPolynomial<23, 16> { using ZPZ = aerobus::zpz<23>; using type =
                            POLYYCZPZVC1>, ZPZVCO>, ZPZVCO
                                                template<> struct ConwayPolynomial<23, 17> { using ZPZ = aerobus::zpz<23>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                            // NOLINT
                            template<> struct ConwayPolynomial<23, 18> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<2>, ZPZV<1>,
04348
                            ZPZV<18>, ZPZV<3>, ZPZV<16>, ZPZV<21>, ZPZV<0>, ZPZV<11>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT
                            template<> struct ConwayPolynomial<23, 19> { using ZPZ = aerobus::zpz<23>; using type POLYV<ZPZV<1>, ZPZV<0>, Z
                                                                                                                                                                                                                                                                                                                                                                                                                                               ZPZV<0>.
                             ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<18»; }; //</pre>
                            NOLINT
04350
                                             template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type =
                            POLYV<ZPZV<1>, ZPZV<27»; }; // NOLINT
04351
                                                template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
                            POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; }; // NOLINT
 04352
                                           template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<27»; }; // NOLINT
template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<29, 5> { using ZPZ = aerobus::zpz<29>; using type =
 04353
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<27»; }; // NOLINT
                                             template<> struct ConwayPolynomial<29, 6> { using ZPZ = aerobus::zpz<29>; using type =
 04355
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<17>, ZPZV<13>, ZPZV<2»; }; // 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>, ZPZV<2>; }; // NOLINT
04356
                                              template<> struct ConwayPolynomial<29, 8> { using ZPZ = aerobus::zpz<29>; using type =
04357
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<2»; };
                            NOLINT
04358
                                           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<2>, ZPZV<22>, ZPZV<27»; };
                             // NOLINT
                                             template<> struct ConwayPolynomial<29, 10> { using ZPZ = aerobus::zpz<29>; using type
04359
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2
                            ZPZV<2»; }; // NOLINT</pre>
04360
                                           template<> struct ConwayPolynomial<29, 11> { using ZPZ = aerobus::zpz<29>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<28>, ZPZV<8>, ZPZV<27»; }; // NOLINT template<> struct ConwayPolynomial<29, 12> { using ZPZ = aerobus::zpz<29>; using type =
04361
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<28>, ZPZV<9>, ZPZV<96>, ZPZV<25>,
                            ZPZV<1>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<29, 13> { using ZPZ = aerobus::zpz<29>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<7
                            template<> struct ConwayPolynomial<29, 14> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<14>, ZPZV<10>,
04363
                            ZPZV<21>, ZPZV<18>, ZPZV<27>, ZPZV<5>, ZPZV<2»; }; // NOLINT
   template<> struct ConwayPolynomial<29, 15> { 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<13>,
                            ZPZV<14>, ZPZV<8>, ZPZV<1>, ZPZV<12>, ZPZV<26>, ZPZV<27»; }; // NOLINT
template<> struct ConwayPolynomial<29, 16> { using ZPZ = aerobus::zpz<29>; using type =
04365
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<2>, ZPZV<18>, ZPZV<23>, ZPZV<1>, ZPZV<27>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<29, 17> { using ZPZ = aerobus::zpz<29>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<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<27»; }; // NOLINT
template<> struct ConwayPolynomial<29, 18> { using ZPZ = aerobus::zpz<29>; using type =
04367
                            POLYY<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<
                                           template<> struct ConwayPolynomial<29, 19> { using ZPZ = aerobus::zpz<29>; using type =
 04368
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<27»; }; //</pre>
                            NOLINT
                                               template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
04369
                          POLYV<ZPZV<1>, ZPZV<28»; }; // NOLINT
                                               template<> struct ConwayPolynomial<31, 2> { using ZPZ = aerobus::zpz<31>; using type =
                          POLYV<ZPZV<1>, ZPZV<29>, ZPZV<3»; }; // NOLINT
                                           template<> struct ConwayPolynomial<31, 3> { using ZPZ = aerobus::zpz<31>; using type =
 04371
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; // NOLINT template<> struct ConwayPolynomial<31, 4> { using ZPZ = aerobus::zpz<31>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3»; };
                                               template<> struct ConwayPolynomial<31, 5> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; }; // NOLINT
                                             template<> struct ConwayPolynomial<31, 6> { using ZPZ = aerobus::zpz<31>; using type =
04374
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<16>, ZPZV<8>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type
04375
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; //
04376
                                               template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>, ZPZV<24>, ZPZV<3»; };
                           template<> struct ConwayPolynomial<31, 9> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<20>, ZPZV<20>, ZPZV<28»; };</pre>
04377
                            // NOLINT
                                                template<> struct ConwayPolynomial<31, 10> { using ZPZ = aerobus::zpz<31>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<26>, ZPZV<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>,
                                               template<> struct ConwayPolynomial<31, 12> { using ZPZ = aerobus::zpz<31>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<14>, ZPZV<28>, ZPZV<2>, ZPZV<9>, ZPZV<25>, ZPZV<12>, ZPZV<3»; }; // NOLINT
                                             template<> struct ConwayPolynomial<31, 13> { using ZPZ = aerobus::zpz<31>; using type =
04381
                            POLYY<ZPZV<1>, 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>,
                            ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<6>, ZPZV<6>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3), ZPZV<12>, ZPZV<12>, ZPZV<13>, ZPZV<13>, ZPZV<25>, ZPZV<28»; }; // NOLINT
template<> struct ConwayPolynomial<31, 16> { using ZPZ = aerobus::zpz<31>; using type =
04384
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             \texttt{ZPZV<24>, \ ZPZV<26>, \ ZPZV<28>, \ ZPZV<11>, \ ZPZV<19>, \ ZPZV<27>, \ ZPZV<3»; \ \}; \ \ // \ \ \texttt{NOLINT} } 
                            template<> struct ConwayPolynomial<31, 17> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04385
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<28»; ); // NOLINT
template<> struct ConwayPolynomial<31, 18> { using ZPZ = aerobus::zpz<31>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2+,
                            ZPZV<2>, ZPZV<7>, ZPZV<12>, ZPZV<11>, ZPZV<25>, ZPZV<25>, ZPZV<10>, ZPZV<6>, ZPZV<6>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type =
04387
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7</pre>, ZPZV<7</pre>
                                                template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
                                                                                                                                                                   // NOLINT
                            POLYV<ZPZV<1>, ZPZV<35»; };
04389
                                             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 =
04390
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
04391
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2»; }; // NOLINT
04392
                                               template<> struct ConwayPolynomial<37, 5> { using ZPZ = aerobus::zpz<37>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35»; }; // NOLINT template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
04393
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<4>, ZPZV<30>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<37, 7> { using ZPZ = aerobus::zpz<37>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>; };
                                               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<27>, ZPZV<29; };
04396
                                             template<> struct ConwayPolynomial<37, 9> { using ZPZ = aerobus::zpz<37>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<20>, ZPZV<32>, ZPZV<35»; };
                             // NOLINT
                                               template<> struct ConwayPolynomial<37, 10> { using ZPZ = aerobus::zpz<37>; using type
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<29>, ZPZV<18>, ZPZV<11>, ZPZV<4>,
                            ZPZV<2»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<37, 11> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
04398
                                                template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type
                            POLYVCZPZVC1>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC4>, ZPZVC31>, ZPZVC10>, ZPZVC23>, ZPZVC23>, ZPZVC18>, ZPZVC33>, ZPZVC2»; }; // NOLINT
                                             template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                template<> struct ConwayPolynomial<37, 14> { using ZPZ = aerobus::zpz<37>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<35>, ZPZV<35>, ZPZV<1>,
                            ZPZV<32>, ZPZV<16>, ZPZV<1>, ZPZV<9>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<37, 15> { using ZPZ = aerobus::zpz<37>; using type =
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>; // NOLINT template<> struct ConwayPolynomial<37, 17> { using ZPZ = aerobus::zpz<37>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<1>, ZPZV<20>, ZPZV<20>, ZPZV<1>, ZPZV<20>, ZPZ
04404
```

```
template<> struct ConwayPolynomial<37, 19> { using ZPZ = aerobus::zpz<37>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<35»; }; //</pre>
                             NOLINT
 04406
                                               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
 04408
                                             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 =
04409
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<6»; }; // NOLINT
                            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
 04410
 04411
                                             template<> struct ConwayPolynomial<41, 6> { using ZPZ = aerobus::zpz<41>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<6>, ZPZV<6 , ZPZV<6
04412
                            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<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<0>, ZPZV<3>, ZPZV<31>, ZPZV<8>, ZPZV<8>, ZPZV<30>,
04415
                             ZPZV<6»; }; // NOLINT
                                            template<> struct ConwayPolynomial<41, 11> { using ZPZ = aerobus::zpz<41>; using type =
04416
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<20>, ZPZV<35»; }; // NOLINT
  template<> struct ConwayPolynomial<41, 12> { using ZPZ = aerobus::zpz<41>; using type =
04417
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26, ZPZV<24>, ZPZV<24>, ZPZV<27, ZPZV<27, ZPZV<6»; }; // NOLINT
                                                template<> struct ConwayPolynomial<41, 13> { using ZPZ = aerobus::zpz<41>; using type =
04418
                             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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<15>, ZPZV<15>, ZPZV<4>,
ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6»; }; // NOLINT
04419
                                                template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             template<> struct ConwayPolynomial<41, 17> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04421
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35s; }; // NOLINT template<> struct ConwayPolynomial<41, 18> { using ZPZ = aerobus::zpz<41>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<23>, ZPZV<35>, ZPZV<38>, ZPZV<24>, ZPZV<12>, ZPZV<29>, ZPZV<10>, ZPZV<6>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<35»; }; //
                             NOLINT
                                                 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 =
 04425
                           POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<43, 3> { using ZPZ = aerobus::zpz<43>; using type =
04426
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40»; }; // NOLINT
                                             template<> struct ConwayPolynomial<43, 4> { using ZPZ = aerobus::zpz<43>; using type =
 04427
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3»; }; // NOLINT
 04428
                                                template<> struct ConwayPolynomial<43, 5> { using ZPZ = aerobus::zpz<43>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<40>, ZPZV<40>, ZPZV<40>; }; // NOLINT template<> struct ConwayPolynomial<43, 6> { using ZPZ = aerobus::zpz<43>; using type =
04429
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<21>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<43, 7> { using ZPZ = aerobus::zpz<43>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<42>, ZPZV<7>, ZPZV<40»; };
 04431
                                            template<> struct ConwayPolynomial<43, 8> { using ZPZ = aerobus::zpz<43>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<39>, ZPZV<20>, ZPZV<24>, ZPZV<3*, }; //
                            NOLINT
04432
                                              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<0>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<40»; };
04433
                                            template<> struct ConwayPolynomial<43, 10> { using ZPZ = aerobus::zpz<43>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<26>, ZPZV<36>, ZPZV<5>, ZPZV<27>, ZPZV<24>,
                             ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type
04434
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<7>, ZPZV<40»; };</pre>
                                                                                                                                        // NOLINT
04435
                                             template<> struct ConwayPolynomial<43, 12> { using ZPZ = aerobus::zpz<43>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<27>, ZPZV<16>, ZPZV<17>, ZPZV<6>,
                             ZPZV<23>, ZPZV<38>, ZPZV<3»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<43, 13> { using ZPZ = aerobus::zpz<43>; using type
04436
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                             ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<40»; };</pre>
                                                                                                                                                                                                                             // NOLINT
                                             template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type =
                             \texttt{POLYV} < \texttt{ZPZV} < \texttt{1}>, \ \texttt{ZPZV} < \texttt{0}>, \ \texttt{ZPZV} < \texttt{3}>, \ \texttt{ZPZV} < \texttt{2}>, \
                            ZPZV<37>, ZPZV<18>, ZPZV<4>, ZPZV<19>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<43, 15> { using ZPZ = aerobus::zpz<43>; using type =
 04438
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZV<3
                              ZPZV<22>, ZPZV<42>, ZPZV<4>, ZPZV<15>, ZPZV<37>, ZPZV<40»; }; // NOLINT
template<> struct ConwayPolynomial<43, 17> { using ZPZ = aerobus::zpz<43>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<40»; }; // NOLINT
template<> struct ConwayPolynomial<43, 18> { using ZPZ = aerobus::zpz<43>; using type =
                             POLYYCZPZVC1>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC1>, ZPZVC3>, ZPZVC28>, ZPZVC41>, ZPZVC24>, ZPZVC7>, ZPZVC24>, ZPZVC24>, ZPZVC3>, Z
                                               template<> struct ConwayPolynomial<43, 19> { using ZPZ = aerobus::zpz<43>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              NOLINT
04442
                                                  template<> struct ConwayPolynomial<47, 1> { using ZPZ = aerobus::zpz<47>; using type =
                              POLYV<ZPZV<1>, ZPZV<42»; }; // NOLINT
 04443
                                               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 =
04444
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42»; };
                                                                                                                                                                                                                                                                  // NOLINT
                                                  template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<47, 5> { using ZPZ = aerobus::zpz<47>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42»; }; // NOLINT
                                                template<> struct ConwayPolynomial<47, 6> { using ZPZ = aerobus::zpz<47>; using type =
04447
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<35>, ZPZV<9>, ZPZV<41>, ZPZV<5»; ); // NOLINT template<> struct ConwayPolynomial<47, 7> { using ZPZ = aerobus::zpz<47>; using type
04448
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT
 04449
                                                template<> struct ConwayPolynomial<47, 8> { using ZPZ = aerobus::zpz<47>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<19>, ZPZV<3>, ZPZV<5»; };
                             template<> struct ConwayPolynomial<47, 9> { using ZPZ = aerobus::zpz<47>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<42»; };</pre>
 04450
                              // 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<45>,
04451
                              ZPZV<5»; }; // NOLINT</pre>
                             \label{template} $$ \text{template}$ <> $$ \text{struct ConwayPolynomial}$ <47, 11> { using $PZ = aerobus:: zpz<47>; using type = $POLYV<ZPZV<1>, $ZPZV<0>, $ZPZV<
 04452
                              ZPZV<6>, ZPZV<42»; };</pre>
                                                                                                                                               // NOLINT
                                                  template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<40>, ZPZV<45>, ZPZV<35>, ZPZV<12>, ZPZV<46>,
                               ZPZV<14>, ZPZV<9>, ZPZV<5»; }; // NOLINT</pre>
04454
                                                template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                  template<> struct ConwayPolynomial<47, 14> { using ZPZ = aerobus::zpz<47>; using type =
04455
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<30>,
                               ZPZV<17>, ZPZV<24>, ZPZV<9>, ZPZV<32>, ZPZV<5»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<47, 15> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04456
04457
                              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                               \texttt{ZPZV} < \texttt{0>, ZPZV} < \texttt{16>, ZPZV} < \texttt{42} \\ \texttt{; }; // \texttt{NOLINT} 
04458
                                               template<> struct ConwayPolynomial<47, 18> { using ZPZ = aerobus::zpz<47>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<41>, ZPZV<42>,
                              ZPZV<26>, ZPZV<44>, ZPZV<24>, ZPZV<22>, ZPZV<11>, ZPZV<5>, ZPZV<45>, ZPZV<33>, ZPZV<5»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<br/>47, 19> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV
04459
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<42»; };</pre>
04460
                                                  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 =
04461
                             POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<53, 3> { using ZPZ = aerobus::zpz<53>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51»; };
                                                                                                                                                                                                                                                                   // NOLINT
                                                 template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
 04463
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<38>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<53, 5> { using ZPZ = aerobus::zpz<53>; using type =
04464
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT
 04465
                                                  template<> struct ConwayPolynomial<53, 6> { using ZPZ = aerobus::zpz<53>; using type =
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<4>, ZPZV<45>, ZPZV<2»; }; // NOLINT
 04466
                                               template<> struct ConwayPolynomial<53, 7> { using ZPZ = aerobus::zpz<53>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5), ZPZV<5), ZPZV<5, ZPZV<5
 04467
                                                  template<> struct ConwayPolynomial<53, 8> { using ZPZ = aerobus::zpz<53>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<1>, ZPZV<2»; };
                                                  template<> struct ConwayPolynomial<53, 9> { using ZPZ = aerobus::zpz<53>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<13>, ZPZV<5>, ZPZV<51»; };
                                                template<> struct ConwayPolynomial<53, 10> { using ZPZ = aerobus::zpz<53>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<29>,
                              ZPZV<2»; }; // NOLINT</pre>
                                                  template<> struct ConwayPolynomial<53, 11> { using ZPZ = aerobus::zpz<53>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<15>, ZPZV<51»; }; // NOLINT</pre>
04471
                                                template<> struct ConwayPolynomial<53, 12> { using ZPZ = aerobus::zpz<53>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<34>, ZPZV<41>, ZPZV<10>, ZPZV<10, ZPZV<42>,
                               ZPZV<34>, ZPZV<41>, ZPZV<2»; }; // NOLINT</pre>
```

```
template<> struct ConwayPolynomial<53, 13> { using ZPZ = aerobus::zpz<53>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<52>, ZPZV<28>, ZPZV<51»; }; // NOLINT
template<> struct ConwayPolynomial<53, 14> { using ZPZ = aerobus::zpz<53>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
                          ZPZV<0>, ZPZV<37>, ZPZV<12>, ZPZV<23>, ZPZV<2»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<53, 15> { using ZPZ = aerobus::zpz<53>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51»; }; // NOLINT
template<> struct ConwayPolynomial<53, 17> { using ZPZ = aerobus::zpz<53>; using type =
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                                           template<> struct ConwayPolynomial<53, 18> { using ZPZ = aerobus::zpz<53>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<52>, ZPZV<51>,
                           ZPZV<27>, ZPZV<0>, ZPZV<39>, ZPZV<44>, ZPZV<6>, ZPZV<8>, ZPZV<16>, ZPZV<11>, ZPZV<2»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<53, 19> { using ZPZ = aerobus::zpz<53>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04477
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<51»; }; //</pre>
                                            template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<57»; }; // NOLINT
                                           template<> struct ConwayPolynomial<59, 2> { using ZPZ = aerobus::zpz<59>; using type =
04479
                          POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
04480
                         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
 04482
                                           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 template<> struct ConwayPolynomial<59, 7> { using ZPZ = aerobus::zpz<59>; using type =
04483
 04484
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<57»; }; // NOLINT
                                        template<> struct ConwayPolynomial<59, 8> { using ZPZ = aerobus::zpz<59>; using type =
 04485
                          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 =
04486
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<1>, ZPZV<32>, ZPZV<47, 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
                                           template<> struct ConwayPolynomial<59, 11> { using ZPZ = aerobus::zpz<59>; using type
04488
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<51>, ZPZV<21>, ZPZV<38>, ZPZV<8>, ZPZV<1>, ZPZV<2»; }; // NOLINT
04490
                                         template<> struct ConwayPolynomial<59, 13> { using ZPZ = aerobus::zpz<59>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57»; };</pre>
                                                                                                                                                                                                       // NOLINT
                                           template<> struct ConwayPolynomial<59, 14> { using ZPZ = aerobus::zpz<59>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<51>, ZPZV<11>,
                          ZPZV<13>, ZPZV<25>, ZPZV<32>, ZPZV<26>, ZPZV<2*, }; // NOLINT
    template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type =
04492
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<24>, ZPZV<23>, ZPZV<23>, ZPZV<39>, ZPZV<58>, ZPZV<57»; }; // NOLINT
                                         template<> struct ConwayPolynomial<59, 17> { using ZPZ = aerobus::zpz<59>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<57»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<37>, ZPZV<38>, ZPZV<27>, ZPZV<11>, ZPZV<14>, ZPZV<14>, ZPZV<16>, ZPZV<47>, ZPZV<44>, ZPZV<38>, ZPZV<28; ; // NOLINT
04494
                         template<> struct ConwayPolynomial<59, 19> { using ZPZ = aerobus::zpz<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<57»; }; //</pre>
                          NOLINT
                                           template<> struct ConwayPolynomial<61, 1> { using ZPZ = aerobus::zpz<61>; using type =
04496
                         POLYV<ZPZV<1>, ZPZV<59»; }; // NOLINT
 04497
                                            template<> struct ConwayPolynomial<61, 2> { using ZPZ = aerobus::zpz<61>; using type =
                          POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; }; // NOLINT
 04498
                                         template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; }; // NOLINT template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
04499
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<61, 5> { using ZPZ = aerobus::zpz<61>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59»; }; // NOLINT
 04501
                                            template<> struct ConwayPolynomial<61, 6> { using ZPZ = aerobus::zpz<61>; using type =
                          \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 4>, \ \texttt{ZPZV} < 3>, \ \texttt{ZPZV} < 2>, \ \texttt{ZPZV} < 2>; \ \}; \ \ // \ \ \texttt{NOLINT} 
                         template<> struct ConwayPolynomial<61, 7> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; }; // NOLINT
04502
                                           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»; };
 04504
                                         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<9>, ZPZV<50>, ZPZV<50>, ZPZV<59»; };
                          // NOLINT
 04505
                                         template<> struct ConwayPolynomial<61, 10> { using ZPZ = aerobus::zpz<61>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<28>, ZPZV<15>, ZPZV<44>, ZPZV<16>, ZPZV<6>,
                         ZPZV<2»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<61, 11> { using ZPZ = aerobus::zpz<61>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<18>, ZPZV<59»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<61, 12> { using ZPZ = aerobus::zpz<61>; using type =
04507
                         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»; };</pre>
                                                                                                                                                               // NOLINT
                                        template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type
04508
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         \texttt{template<>} \ \texttt{struct ConwayPolynomial<61, 14> \{ \ using \ \texttt{ZPZ = aerobus::zpz<61>; using \ \texttt{type} \} \} }
04509
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<48>, ZPZV<48>, ZPZV<26>, ZPZV<11>,
                         ZPZV<8>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<2»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<61, 15> { using ZPZ = aerobus::zpz<61>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<35>, ZPZV<44>, ZPZV<25>, ZPZV<23>, ZPZV<51>, ZPZV<59»; }; // NOLINT
    template<> struct ConwayPolynomial<61, 17> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04511
                         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<3>, ZPZV<35>, ZPZV<36>, ZPZV<13>,
                         ZPZV<36>, ZPZV<4>, ZPZV<32>, ZPZV<35>, ZPZV<42>, ZPZV<25>, ZPZV<25>, ZPZV<25>, ZPZV<22>; }; // NOLINT template<> struct ConwayPolynomial<61, 19> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<
04513
04514
                                          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 =
04515
                         POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
04516
                                           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 =
04517
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<54>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<67, 5> { using ZPZ = aerobus::zpz<67>; using type =
04518
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<67, 6> { using ZPZ = aerobus::zpz<67>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<63>, ZPZV<49>, ZPZV<55>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<67, 7> { using ZPZ = aerobus::zpz<67>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<65»; };
04521
                                         template<> struct ConwayPolynomial<67, 8> { using ZPZ = aerobus::zpz<67>; using type =
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<46>, ZPZV<17>, ZPZV<64>, ZPZV<62»; };
                         template<> struct ConwayPolynomial<67, 9> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<55>, ZPZV<55>; ZPZV<65»; };</pre>
                         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>,
04523
                         ZPZV<2»: }: // NOLINT</pre>
                                          template<> struct ConwayPolynomial<67, 11> { using ZPZ = aerobus::zpz<67>; using type
04524
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<9>, ZPZV<65»; }; // NOLINT
template<>> struct ConwayPolynomial<67, 12> { using ZPZ = aerobus::zpz<67>; using type =
04525
                         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<2»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<65»; }; // NOLINT</pre>
                         template<> struct ConwayPolynomial<67, 14> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<22>, ZPZV<55>, ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<37>, ZPZV<22>; }; // NOLINT template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type =
04527
                         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>
                         \label{eq:convayPolynomial} $$ \text{template} <> \text{struct ConwayPolynomial} <67, 17> $$ using $$ \text{ZPZ} = \frac{\text{aerobus}::} 2pz <67>$$; using type = $$ POLYV < 2pz V <0>, $$ ZPZ V <0>, $$
04529
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<6>; // NOLINT template<> struct ConwayPolynomial<67, 18> { using ZPZ = aerobus::zpz<67>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<63>, ZPZV<63>, ZPZV<63>, ZPZV<52>, ZPZV<18>,
                          ZPZV<33>, ZPZV<55>, ZPZV<28>, ZPZV<29>, ZPZV<51>, ZPZV<6>, ZPZV<59>, ZPZV<13>, ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<67, 19> { using ZPZ = aerobus::zpz<67>; using type =
04531
                         POLYYCZPZVC1>, ZPZVC0>, ZPZVC0
                         NOLINT
                                           template<> struct ConwayPolynomial<71, 1> { using ZPZ = aerobus::zpz<71>; using type =
                         POLYV<ZPZV<1>, ZPZV<64»; }; // NOLINT
                                          template<> struct ConwayPolynomial<71, 2> { using ZPZ = aerobus::zpz<71>; using type =
                        POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7»; }; // NOLINT
                                          template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
04534
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; }; // NOLINT template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
04535
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<71, 5> { using ZPZ = aerobus::zpz<71>; using type =
04536
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<64»; }; // NOLINT template<> struct ConwayPolynomial<71, 6> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<13>, ZPZV<29>, ZPZV<7»; }; // NOLINT
04537
```

```
template<> struct ConwayPolynomial<71, 7> { using ZPZ = aerobus::zpz<71>; using type
                         template<> struct ConwayPolynomial<71, 8> { using ZPZ = aerobus::zpz<71>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<22>, ZPZV<19>, ZPZV<7»; };
                          NOLINT
                                         template<> struct ConwayPolynomial<71, 9> { using ZPZ = aerobus::zpz<71>; using type =
04540
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43>, ZPZV<43>, ZPZV<62>, ZPZV<64»; };
                                          template<> struct ConwayPolynomial<71, 10> { using ZPZ = aerobus::zpz<71>; using type =
04541
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<15>, ZPZV<16>, ZPZV<26>, ZPZV<10>, ZPZV<40>,
                          ZPZV<7»: }: // NOLINT
                                         template<> struct ConwayPolynomial<71, 11> { using ZPZ = aerobus::zpz<71>; using type =
04542
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        template<> struct ConwayPolynomial<71, 12> { using ZPZ = aerobus::zpz<71>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<28>, ZPZV<29>, ZPZV<55>, ZPZV<21>, ZPZV<58>, ZPZV<23>, ZPZV<7»; }; // NOLINT
                                         template<> struct ConwayPolynomial<71, 13> { using ZPZ = aerobus::zpz<71>; using type =
04544
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<32>, ZPZV<18>, ZPZV<52>, ZPZV<67>, ZPZV<49>, ZPZV<64»; }; // NOLINT
  template<> struct ConwayPolynomial<71, 17> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04546
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<64»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<71, 19> { using ZPZ = aerobus::zpz<71>; using type
04547
                                                                                                                                                                                                                                                                                                                                                                                                        ZPZV<0>,
                           \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ 
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<64»; }; //</pre>
                          NOLINT
                                         template<> struct ConwayPolynomial<73, 1> { using ZPZ = aerobus::zpz<73>; using type =
04548
                         POLYV<ZPZV<1>, ZPZV<68»; }; // NOLINT
                                           template<> struct ConwayPolynomial<73, 2> { using ZPZ = aerobus::zpz<73>; using type =
                         POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5»; }; // NOLINT
 04550
                                           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 =
04551
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5»; }; // NOLINT
 04552
                                           template<> struct ConwayPolynomial<73, 5> { using ZPZ = aerobus::zpz<73>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<68»; }; // NOLINT
 04553
                                        template<> struct ConwayPolynomial<73, 6> { using ZPZ = aerobus::zpz<73>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<45>, ZPZV<23>, ZPZV<48>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<73, 7> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
04554
                                           template<> struct ConwayPolynomial<73, 8> { using ZPZ = aerobus::zpz<73>; using type :
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, 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>
04556
                          // NOLINT
04557
                                           template<> struct ConwayPolynomial<73, 10> { using ZPZ = aerobus::zpz<73>; using type =
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<23>, ZPZV<33>, ZPZV<33>, ZPZV<32>, ZPZV<69>,
                          ZPZV<5»; }; // NOLINT</pre>
04558
                                          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<1>, ZPZV<69>, ZPZV<52>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<46>,
                          ZPZV<29>, ZPZV<25>, ZPZV<5»; }; // NOLINT</pre>
04560
                                           template<> struct ConwayPolynomial<73, 13> { using ZPZ = aerobus::zpz<73>; using type
                         POLYY<ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68»; }; // NOLINT
                                           template<> struct ConwayPolynomial<73, 15> { using ZPZ = aerobus::zpz<73>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         template<> struct ConwayPolynomial<73, 17> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6 ,
04562
04563
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68»; }; //</pre>
04564
                                           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 =
04565
                         POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3»; }; // NOLINT
                                         template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76»; }; // NOLINT template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
 04567
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<79, 5> { using ZPZ = aerobus::zpz<79>; using type =
 04568
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<76»; }; // NOLINT
                                           template<> struct ConwayPolynomial<79, 6> { using ZPZ = aerobus::zpz<79>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<68>, ZPZV<3»; }; // NOLINT
                       template<> struct ConwayPolynomial<79, 7> { using ZPZ = aerobus::zpz<79>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76»; }; // NOLINT
template<> struct ConwayPolynomial<79, 8> { using ZPZ = aerobus::zpz<79>; using type =
 04570
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<60>, ZPZV<59>, ZPZV<48>, ZPZV<48>; };
                                         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<19>, ZPZV<76»; };
                           // NOLINT
                                            template<> struct ConwayPolynomial<79, 10> { using ZPZ = aerobus::zpz<79>; using type =
04573
                           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 =
04574
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<3>, ZPZV<76»; }; // NOLINT
                                           template<> struct ConwayPolynomial<79, 12> { using ZPZ = aerobus::zpz<79>; using type
04575
                          POLYYCZPZVC1>, ZPZVC3>, ZPZVC3>, ZPZVC4>, ZPZVC4>, ZPZVC4>, ZPZVC4>, ZPZVC4>, ZPZVC4>, ZPZVC4>, ZPZVC5>, ZPZVC5>, ZPZVC3>; // NOLINT
                                         template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type
04577
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<76»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<79, 19> { using ZPZ = aerobus::zpz<79>; using type POLYV<ZPZV<1>, ZPZV<0>, Z
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<76»; }; //</pre>
                           NOLINT
04579
                                            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 =
04580
                          POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; }; // NOLINT
04581
                                            template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT template<> struct ConwayPolynomial<83, 4> { using ZPZ = aerobus::zpz<83>; using type =
04582
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2»; };
                                                                                                                                                                                                                                                                                     // NOLINT
                                            template<> struct ConwayPolynomial<83, 5> { using ZPZ = aerobus::zpz<83>; using type =
04583
                           \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<9>, \verb"ZPZV<81"; \verb"}; \verb"// NOLINT" | 
04584
                                            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::2pz<83>, using type
04585
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT
                                            template<> struct ConwayPolynomial<83, 8> { using ZPZ = aerobus::zpz<83>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<23>, ZPZV<42>, ZPZV<22»; };
04587
                         template<> struct ConwayPolynomial<83, 9> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<81»; };</pre>
                           // NOLINT
                                            template<> struct ConwayPolynomial<83, 10> { using ZPZ = aerobus::zpz<83>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5 , ZPZV<5
                           ZPZV<2»; }; // NOLINT</pre>
                          template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                           ZPZV<17>, ZPZV<81»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type =
04590
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35>, ZPZV<12>, ZPZV<31>, ZPZV<19>, ZPZV<65>, ZPZV<55>, ZPZV<75>, ZPZV<2»; }; // NOLINT
04591
                                            template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<81»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<83, 17> { using ZPZ = aerobus::zpz<83>; using type
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<81»; };</pre>
04593
                                             template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<47>, ZPZV<41>; //
                           NOLINT
                                             template<> struct ConwayPolynomial<89, 1> { using ZPZ = aerobus::zpz<89>; using type =
                          POLYV<ZPZV<1>, ZPZV<86»; }; // NOLINT
04595
                                           template<> struct ConwayPolynomial<89, 2> { using ZPZ = aerobus::zpz<89>; using type =
                          POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; }; // NOLINT
                                           template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
04596
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<86»; }; // NOLINT
                                             template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
04597
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<89, 5> { using ZPZ = aerobus::zpz<89>; using type =
04598
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<86»; }; // NOLINT template<> struct ConwayPolynomial<89, 6> { using ZPZ = aerobus::zpz<89>; using type =
04599
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<80>, ZPZ
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<86»; };
04601
                                          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
04602
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<12>, ZPZV<6>, ZPZV<86»; };
04603
                                           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»; }; // NOLINT
04604
                                          template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     ZPZV<26>, ZPZV<86»; }; // NOLINT</pre>
                                                          template<> struct ConwayPolynomial<89, 12> { using ZPZ = aerobus::zpz<89>; using type =
                                    POLYY<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
                                                             template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type =
 04606
                                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                          template<> struct ConwayPolynomial<89, 17> { using ZPZ = aerobus::zpz<89>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
 04608
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 7P7V<0>.
                                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<86»; }; //</pre>
                                     NOLINT
 04609
                                                             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 =
 04610
                                     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 =
  04612
                                  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
  04613
                                                              template<> struct ConwayPolynomial<97, 6> { using ZPZ = aerobus::zpz<97>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<92>, ZPZV<58>, ZPZV<88>, ZPZV<5»; }; // NOLINT
                                                           template<> struct ConwayPolynomial<97, 7> { using ZPZ = aerobus::zpz<97>; using type =
  04615
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<92»; };
                                                          template<> struct ConwayPolynomial<97, 8> { using ZPZ = aerobus::zpz<97>; using type =
 04616
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<65, ZPZV<6
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59>, ZPZV<7>, ZPZV<92»; };
                                     // NOLINT
                                    template<> struct ConwayPolynomial<97, 10> { using ZPZ = 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>,
 04618
                                     ZPZV<5»; }; // NOLINT
                                                             template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<5>, ZPZV<92»; };</pre>
                                                                                                                                                                              // NOLINT
 04620
                                                          template<> struct ConwayPolynomial<97, 12> { using ZPZ = aerobus::zpz<97>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<59>, ZPZV<81>, ZPZV<86>, ZPZV<78>, ZPZV<94>, ZPZV<59>; // NOLINT
 04621
                                                             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
                                                          template<> struct ConwayPolynomial<97, 17> { using ZPZ = aerobus::zpz<97>; using type =
04622
                                    Template<> Struct ConwayPolynomials97, 1/> { using ZrZ = derobus::ZpZ<9/>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
 04623
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<92»; }; //</pre>
 04624
                                                             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 =
 04625
                                     POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; }; // NOLINT
                                                          template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
                                    POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<3>, ZPZY<3>; // NOLINT template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
  04627
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<78, ZPZV<2»; }; // NOLINT

template<> struct ConwayPolynomial(101, 5> { using ZPZ = aerobus::zpz<101>; using type =
 04628
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<99»; }; // NOLINT
                                                             template<> struct ConwayPolynomial<101, 6> { using ZPZ = aerobus::zpz<101>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<90>, ZPZV<20>, ZPZV<67>, ZPZV<2»; }; // NOLINT
                                                          template<> struct ConwayPolynomial<101, 7> { using ZPZ = aerobus::zpz<101>; using type =
  04630
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
 04631
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76>, ZPZV<29>, ZPZV<24>, ZPZV<24>, ZPZV<29, ZPZV<24>, ZPZV<25, ZPZV<29, ZPZV<24>, ZPZV<25, ZPZV<25, ZPZV<25, ZPZV<25, ZPZV<25, ZPZV<25, ZPZV<26, ZPZV<26, ZPZV<27, ZPZV<28, 
                                    NOLINT
                                                             template<> struct ConwayPolynomial<101, 9> { using ZPZ = aerobus::zpz<101>; using type =
 04632
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64>, ZPZV<47>, ZPZV<99»; };
                                        // NOLINT
 04633
                                                           template<> struct ConwayPolynomial<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
                                                          template<> struct ConwayPolynomial<101, 12> { using ZPZ = aerobus::zpz<101>; using type =
 04635
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<79>, ZPZV<64>, ZPZV<39>, ZPZV<78>, ZPZV<48>,
                                     ZPZV<84>, ZPZV<21>, ZPZV<2»; }; // NOLINT</pre>
                                                          template<> struct ConwayPolynomial<101, 13> { using ZPZ = aerobus::zpz<101>; using type
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     \text{ZPZV}<0>, \text{ZPZV}<0>, \text{ZPZV}<7>, \text{ZPZV}<99»; }; // NOLINT
                                    template<> struct ConwayPolynomial<101, 17> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<99»; };</pre>
                      template<> struct ConwayPolynomial<101, 19> { using ZPZ = aerobus::zpz<101>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<9»; }; //</pre>
                       NOLINT
04639
                                      template<> struct ConwavPolynomial<103, 1> { using ZPZ = aerobus::zpz<103>; using type =
                      POLYV<ZPZV<1>, ZPZV<98»; }; // NOLINT
04640
                                       template<> struct ConwayPolynomial<103, 2> { using ZPZ = aerobus::zpz<103>; using type =
                      POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; }; // NOLINT
04641
                                     template<> struct ConwayPolynomial<103, 3> { using ZPZ = aerobus::zpz<103>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<98»; }; // NOLINT
                                     template<> struct ConwayPolynomial<103, 4> { using ZPZ = aerobus::zpz<103>; using type =
04642
                      POLYYCZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<88>, ZPZV<5>; }; // NOLINT template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
04643
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<98»; }; // NOLINT
04644
                                      template<> struct ConwayPolynomial<103, 6> { using ZPZ = aerobus::zpz<103>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<96>, ZPZV<9>, ZPZV<30>, ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<103, 7> { using ZPZ = aerobus::zpz<103>; using type =
04645
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98»; }; // NOLINT
                                     template<> struct ConwayPolynomial<103, 8> { using ZPZ = aerobus::zpz<103>; using type =
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<71>, ZPZV<71>, ZPZV<49>, ZPZV<5»; };
                       NOLINT
04647
                                     template<> struct ConwayPolynomial<103, 9> { using ZPZ = aerobus::zpz<103>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<51>, ZPZV<58*; };
                        // NOLINT
                                       template<> struct ConwayPolynomial<103, 10> { using ZPZ = aerobus::zpz<103>; using type :
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<101>, ZPZV<86>, ZPZV<101>, ZPZV<94>, ZPZV<11>,
                        ZPZV<5»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<103, 11> { using ZPZ = aerobus::zpz<103>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04650
                                       template<> struct ConwayPolynomial<103, 12> { using ZPZ = aerobus::zpz<103>; using type
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<25, ZPZV<81>, ZPZV<29, ZPZV<88>, ZPZV<88>, ZPZV<5»; }; // NOLINT
                      template<> struct ConwayPolynomial<103, 13> { using ZPZ = aerobus::zpz<103>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
04651
                                      template<> struct ConwayPolynomial<103, 17> { using ZPZ = aerobus::zpz<103>; using type
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<102>, ZPZV<8>, ZPZV<98»; }; // NOLINT</pre>
04653
                                    template<> struct ConwayPolynomial<103, 19> { using ZPZ = aerobus::zpz<103>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3</pre>
                                       template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
                      POLYV<ZPZV<1>, ZPZV<105»; }; // NOLINT
04655
                                    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 =
04656
                      POLYVCZPZV<1>, ZPZV<5>, ZPZV<55, ZPZV<105»; }; // NOLINT template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
04657
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; }; // NOLINT
04658
                                      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 =
04659
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<52>, ZPZV<7>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<105»; };
                                    template<> struct ConwayPolynomial<107, 8> { using ZPZ = aerobus::zpz<107>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<24>, ZPZV<95>, ZPZV<95; };
                      NOLINT
                                     template<> struct ConwayPolynomial<107, 9> { using ZPZ = aerobus::zpz<107>; using type =
04662
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<66>, 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<85>,
                       ZPZV<2»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<107, 11> { using ZPZ = aerobus::zpz<107>; using type =
04664
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<8>, ZPZV<105»; };</pre>
                                                                                                                  // NOLINT
                                      template<> struct ConwayPolynomial<107, 12> { using ZPZ = aerobus::zpz<107>; using type :
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<48>, ZPZV<6>, ZPZV<6>, ZPZV<61>,
                       ZPZV<42>, ZPZV<57>, ZPZV<2»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type :
04666
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    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<2>, ZPZV<105»; }; // NOLINT
template<> struct ConwayPolynomial<107, 19> { using ZPZ = aerobus::zpz<107>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04668
                        ZPZV<0>, ZPZV<2+, ZPZV<10*, }; //</pre>
04669
                                     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 =
04670
                       POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; }; // NOLINT
```

```
04671
                                         template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
                       POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<103»; }; // NOLINT template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
04672
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; }; // NOLINT
                                         template<> struct ConwayPolynomial<109, 5> { using ZPZ = aerobus::zpz<109>; using type =
 04673
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103»; }; // NOLINT
                                         template<> struct ConwayPolynomial<109, 6> { using ZPZ = aerobus::zpz<109>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<102>, ZPZV<66>, ZPZV<66»; }; // NOLINI
 04675
                                      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<1>, ZPZV<14>, ZPZV<103»; };
                                       template<> struct ConwayPolynomial<109, 8> { using ZPZ = aerobus::zpz<109>; using type =
04676
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<34>, ZPZV<86>; };
                                         template<> struct ConwayPolynomial<109, 9> { using ZPZ = aerobus::zpz<109>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5,
                                       template<> struct ConwayPolynomial<109, 10> { using ZPZ = aerobus::zpz<109>; using type =
04678
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<71>, ZPZV<55>, ZPZV<16>, ZPZV<75>, ZPZV<69>,
                         ZPZV<6»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<109, 11> { using ZPZ = aerobus::zpz<109>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<11>, ZPZV<103»; }; // NOLINT
                                       template<> struct ConwayPolynomial<109, 12> { using ZPZ = aerobus::zpz<109>; using type =
04680
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<50>, ZPZV<53>, ZPZV<37>, ZPZV<37>, ZPZV<85>,
                         ZPZV<103>, ZPZV<28>, ZPZV<6»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<109, 13> { using ZPZ = aerobus::zpz<109>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<103»; };</pre>
                                                                                                                                                                                            // NOLINT
04682
                                         template<> struct ConwayPolynomial<109, 17> { using ZPZ = aerobus::zpz<109>; using type
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
04683
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<103»; }; //</pre>
                         NOLINT
                                         template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
04684
                        POLYV<ZPZV<1>, ZPZV<110»; }; // NOLINT
                                         template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
                        POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3»; }; // NOLINT
                                       template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<110»; }; // NOLINT template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
04687
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<62>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<113, 5> { using ZPZ = aerobus::zpz<113>; using type =
04688
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110»; }; // NOLINT
 04689
                                         template<> struct ConwayPolynomial<113, 6> { using ZPZ = aerobus::zpz<113>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<59>, ZPZV<30>, ZPZV<71>, ZPZV<3»; }; // NOLINT
04690
                                       template<> struct ConwayPolynomial<113, 7> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZV<0>, ZPZV<1>; // NOLINT template<> struct ConwayPolynomial<113, 8> { using ZPZ = aerobus::zpz<113>; using type =
04691
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<98>, ZPZV<38>, ZPZV<28>, ZPZV<3»; };
04692
                                      template<> struct ConwayPolynomial<113, 9> { using ZPZ = aerobus::zpz<113>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<87>, ZPZV<71>, ZPZV<110»; };
                         // NOLINT
                        template<> struct ConwayPolynomial<113, 10> { using ZPZ = aerobus::zpz<113>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<108>, ZPZV<57>, ZPZV<45>, ZPZV<83>, ZPZV<56>,
04693
                         ZPZV<3»: }: // NOLINT
                                       template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                         ZPZV<3>, ZPZV<110»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<113, 12> { using ZPZ = aerobus::zpz<113>; using type =
04695
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<50>, ZPZ
                                      template<> struct ConwayPolynomial<113, 13> { using ZPZ = aerobus::zpz<113>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110»; };</pre>
                                                                                                                                                                                            // NOLINT
                                       template<> struct ConwayPolynomial<113, 17> { using ZPZ = aerobus::zpz<113>; using type =
04697
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110»; };</pre>
                                                                                                                                                                                                                                                                                                                                                 // NOLINT
                                        template<> struct ConwayPolynomial<113, 19> { using ZPZ = aerobus::zpz<113>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<110»; }; //</pre>
                         NOLINT
04699
                                         template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
                        POLYV<ZPZV<1>, ZPZV<124»; }; // NOLINT
 04700
                                       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 =
 04701
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124»; }; // NOLINT template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type =
 04702
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<97>, ZPZV<3»; };
                                                                                                                                                                                                                                                       // NOLINT
                                         template<> struct ConwayPolynomial<127, 5> { using ZPZ = aerobus::zpz<127>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<124»; }; // NOLINT
 04704
                                      template<> struct ConwayPolynomial<127, 6> { using ZPZ = aerobus::zpz<127>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<84>, ZPZV<115>, ZPZV<82>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<127, 7> { using ZPZ = aerobus::zpz<127>; using type =
 04705
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<15>, ZPZV<124»; };
                                  template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<104>, ZPZV<55>, ZPZV<8>, ZPZV<3»; }; //
                    NOLINT
04707
                                  template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12+, ZPZV<14>, ZPZV<119>, ZPZV<126>, ZPZV<124»;
                                 template<> struct ConwayPolynomial<127, 10> { using ZPZ = aerobus::zpz<127>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<64>, ZPZV<95>, ZPZV<60>, ZPZV<4>,
                     ZPZV<3»; }; // NOLINT</pre>
04709
                                 template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<11>, ZPZV<124»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<127, 12> { using ZPZ = aerobus::zpz<127>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<119>, ZPZV<25>, ZPZV<33>, ZPZV<97>, ZPZV<15>,
                      ZPZV<99>, ZPZV<8>, ZPZV<3»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type =
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                                   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>,
                                                                                                                                                                                                                                                                                                     // NOLINT
                     template<> struct ConwayPolynomial<127, 19> { using ZPZ = aerobus::zpz<127>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                                  template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
                    POLYV<ZPZV<1>, ZPZV<129»; }; // NOLINT
04715
                                   template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
                    POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2»; }; // NOLINT
                                  template<> struct ConwayPolynomial<131, 3> { using ZPZ = aerobus::zpz<131>; using type =
04716
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<129»; }; // NOLINT
04717
                                   template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2»; }; // NOLINT
04718
                                   template<> struct ConwayPolynomial<131, 5> { using ZPZ = aerobus::zpz<131>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<129»; }; // NOLINT
                                  template<> struct ConwayPolynomial<131, 6> { using ZPZ = aerobus::zpz<131>; using type =
04719
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<4>, ZPZV<22>, ZPZV<2»; }; // NOLINT
04720
                                   template<> struct ConwayPolynomial<131, 7> { using ZPZ = aerobus::zpz<131>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<129»; }; //
04721
                                 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
04722
                                  template<> struct ConwayPolynomial<131, 9> { using ZPZ = aerobus::zpz<131>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<19>, ZPZV<129»; };
                      // NOLINT
04723
                                  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
                                   template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type
04724
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<6>, ZPZV<129»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<131, 12> { using ZPZ = aerobus::zpz<131>; using type :
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<122>, ZPZV<40>, ZPZV<83>, ZPZV<125>, ZPZV<28>, ZPZV<203>, ZPZV<203
                                  template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type =
04726
                    POLYVCZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<129»; };</pre>
                                                                                                                                                                        // NOLINT
                                   template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<129w; }; // NOLINT
    template<> struct ConwayPolynomial<131, 19> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<0>, 
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<129»; }; //</pre>
04729
                                 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 =
04730
                   POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3»; }; // NOLINT
                                    template<> struct ConwayPolynomial</137, 3> { using ZPZ = aerobus::zpz<137>; using type =
04731
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
04732
                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<95>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
04733
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<73, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 6> { using ZPZ = aerobus::zpz<137>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<116>, ZPZV<102>, ZPZV<3>, ZPZV<3»; };
                   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<134w; }; // NOLINT template<> struct ConwayPolynomial<137, 8> { using ZPZ = aerobus::zpz<137>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3w; }; //
04736
                                 template<> struct ConwayPolynomial<137, 9> { using ZPZ = aerobus::zpz<137>; using type =
                    POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<80>, ZPZV<122>, ZPZV<134»;
                    }; // NOLINT
                    template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<20>, ZPZV<67>, ZPZV<93>, ZPZV<119>,
04738
```

```
ZPZV<3»; };</pre>
                                                     template<> struct ConwayPolynomial<137, 11> { using ZPZ = aerobus::zpz<137>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<1>, ZPZV<134»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<137, 12> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<61>, ZPZV<40>, ZPZV<40>, ZPZV<12>, ZPZV<36>,
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>, ZPZV<0>, ZPZV<14>, ZPZV<134»; }; // NOLINT
template<> struct ConwayPolynomial<137, 17> { using ZPZ = aerobus::zpz<137>; using type =
04742
                                 POLYV<PPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<4>, ZPZV<134; }; // NOLINT
template<> struct ConwayPolynomial<137, 19> { using ZPZ = aerobus::zpz<137>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<134»; }; //</pre>
                                  NOLINT
                                                      template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
04744
                                 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
04746
                                                     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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2»; }; // NOLINT
 04747
                                                       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
 04749
                                                     template<> struct ConwayPolynomial<139, 6> { using ZPZ = aerobus::zpz<139>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<46>, ZPZV<10>, ZPZV<118>, ZPZV<2»; }; // NOLINT
                               template<> struct ConwayPolynomial<139, 7> { using ZPZ = aerobus::zpz<139>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 
04750
 04751
                                                       template<> struct ConwayPolynomial<139, 8> { using ZPZ = aerobus::zpz<139>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>, ZPZV<36>, ZPZV<21>, ZPZV<22, };
                                template<> struct ConwayPolynomial<139, 9> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<70>, ZPZV<70>, ZPZV<87>, ZPZV<137»; };</pre>
04752
                                   // NOLINT
                                                       template<> struct ConwayPolynomial<139, 10> { using ZPZ = aerobus::zpz<139>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0, ZPZV<110>, ZPZV<48>, ZPZV<130>, ZPZV<66>,
                                   ZPZV<106>, ZPZV<2»; };</pre>
                                                                                                                                                              // NOLINT
04754
                                                    template<> struct ConwayPolynomial<139, 11> { using ZPZ = aerobus::zpz<139>; using type =
                                  \texttt{POLYV} < \texttt{ZPZV} < \texttt{0} >, \ \texttt{ZPZV} < \texttt{
                                  ZPZV<7>, ZPZV<137»; }; // NOLINT</pre>
                                                      template<> struct ConwayPolynomial<139, 12> { using ZPZ = aerobus::zpz<139>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<100, ZPZV
04756
                                                   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 =
04757
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137»; };</pre>
04758
                                                   template<> struct ConwayPolynomial<139, 19> { using ZPZ = aerobus::zpz<139>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<137»; }; //</pre>
                                  NOLINT
                                                        template<> struct ConwayPolynomial<149, 1> { using ZPZ = aerobus::zpz<149>; using type =
                                POLYV<ZPZV<1>, ZPZV<147»; }; // NOLINT
                                                       template<> struct ConwayPolynomial<149, 2> { using ZPZ = aerobus::zpz<149>; using type =
                                 POLYV<ZPZV<1>, ZPZV<145>, ZPZV<2»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<149, 3> { using ZPZ = aerobus::zpz<149>; using type =
04761
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
 04762
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<149, 5> { using ZPZ = aerobus::zpz<149>; using type =
04763
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<147»; }; // NOLINT
 04764
                               template<> struct ConwayPolynomial<149, 6> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>>, ZPZV<33>, ZPZV<55>, ZPZV<2»; }; // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<149, 7> { using ZPZ = aerobus::zpz<149>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<147»; }; //
                                                     template<> struct ConwayPolynomial<149, 8> { using ZPZ = aerobus::zpz<149>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<140>, ZPZV<25>, ZPZV<123>, ZPZV<2»; }; //
                                 NOLINT
04767
                                                     template<> struct ConwayPolynomial<149, 9> { using ZPZ = aerobus::zpz<149>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<146>, ZPZV<20>, ZPZV<147»;
                                 }; // NOLINT template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>,
                                  ZPZV<2»; }; // NOLINT</pre>
04769
                                                   template<> struct ConwayPolynomial<149, 11> { using ZPZ = aerobus::zpz<149>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<33>, ZPZV<147»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<91>, ZPZV<92>, ZPZV<9>,
                                 \label{eq:zpzv<104>, zpzv<110>, zpzv<2»; }; // \mbox{NOLINT}
                                 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>
                                                  template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<29>, ZPZV<147»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<149, 19> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5</pre>, //
                                                   template<> struct ConwayPolynomial<151, 1> { using ZPZ = aerobus::zpz<151>; using type =
04774
                             POLYV<ZPZV<1>, ZPZV<145»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<151, 2> { using ZPZ = aerobus::zpz<151>; using type =
04775
                              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
04777
                                                template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<151, 5> { using ZPZ = aerobus::zpz<151>; using type =
04778
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<145»; }; // NOLINT template<> struct ConwayPolynomial<151, 6> { using ZPZ = aerobus::zpz<151>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<125>, ZPZV<18>, ZPZV<15>, ZPZV<6»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<151, 7> { using ZPZ = aerobus::zpz<151>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<145»; }; // NOLINT
04781
                                                 template<> struct ConwayPolynomial<151, 8> { using ZPZ = aerobus::zpz<151>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<140>, ZPZV<122>, ZPZV<43>, ZPZV<6»; }; //
                              NOLINT
                                                   template<> struct ConwayPolynomial<151, 9> { using ZPZ = aerobus::zpz<151>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<125>, ZPZV<96>, ZPZV<96 , ZPZV<97 ,
                               }; // NOLINT
04783
                                                    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
                                                    template<> struct ConwayPolynomial<151, 11> { using ZPZ = aerobus::zpz<151>; using type
                               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<1>, ZPZV<145»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<151, 12> { using ZPZ = aerobus::zpz<151>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<121>, ZPZV<101>, ZPZV<101>, ZPZV<6>, ZPZV<7>,
04785
                               ZPZV<107>, ZPZV<147>, ZPZV<6»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<151, 13> { using ZPZ = aerobus::zpz<151>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<145»; }; // NOLINT</pre>
04787
                                                template<> struct ConwayPolynomial<151,
                                                                                                                                                                                                                                                               17> { using ZPZ = aerobus::zpz<151>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>
                               NOLINT
04789
                                                 template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
                             POLYV<ZPZV<1>, ZPZV<152»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
04790
                             POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5»; }; // NOLINT
                                                   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 =
04792
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<1357, dusing type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<157, ZPZV<1369, ZPZV<39; }; // NOLINT template<> struct ConwayPolynomial<157, 5> { using ZPZ = aerobus::zpz<157>; using type =
04793
                               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 =
04794
                              POLYY<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 =
04795
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; }; // NOLINT
                                                template<> struct ConwayPolynomial<157, 8> { using ZPZ = aerobus::zpz<157>; using type =
04796
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<97>, ZPZV<40>, ZPZV<453>, ZPZV<55; };
04797
                                               template<> struct ConwayPolynomial<157, 9> { using ZPZ = aerobus::zpz<157>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<114>, ZPZV<52>, ZPZV<152»;
                               }; // NOLINT
04798
                                                   template<> struct ConwavPolynomial<157, 10> { using ZPZ = aerobus::zpz<157>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<61>, ZPZV<22>, ZPZV<124>, ZPZV<61>, ZPZV<93>,
                               ZPZV<5»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<157, 11> { using ZPZ = aerobus::zpz<157>; using type :
04799
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<29>, ZPZV<152»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<157, 12> { using ZPZ = aerobus::zpz<157>; using type =
04800
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<110>, ZPZV<72>, ZPZV<137>, ZPZV<43>, ZPZV<152>, ZPZV<57>, ZPZV<55>; }; // NOLINT
04801
                                                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 =
04802
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<152»; }; // NOLINT
template<> struct ConwayPolynomial<157, 19> { using ZPZ = aerobus::zpz<157>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; }; //</pre>
                               NOLINT
04804
                                                  template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<161»; };
                                       template<> struct ConwayPolynomial<163, 2> { using ZPZ = aerobus::zpz<163>; using type =
                       POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
 04806
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<161»; }; // NOLINT template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
04807
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2»; }; // NOLINT
 04808
                                       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 =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<25>, ZPZV<156>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<163, 7> { using ZPZ = aerobus::zpz<163>; using type =
 04809
 04810
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<161»; };
                                       template<> struct ConwayPolynomial<163, 8> { using ZPZ = aerobus::zpz<163>; using type
 04811
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<132>, ZPZV<83>, ZPZV<6>, ZPZV<2»; };
                                     template<> struct ConwayPolynomial<163, 9> { using ZPZ = aerobus::zpz<163>; using type =
04812
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<162>, ZPZV<161>, ZPZV<161»;
                                        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 =
04814
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<11>, ZPZV<161»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<112>, ZPZV<31>, ZPZV<38>, ZPZV<103>,
                        ZPZV<10>, ZPZV<69>, ZPZV<2»; }; // NOLINT</pre>
04816
                                       template<> struct ConwayPolynomial<163, 13> { using ZPZ = aerobus::zpz<163>; using type :
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04817
                                       template<> struct ConwayPolynomial<163,
                                                                                                                                                                                              17> { using ZPZ = aerobus::zpz<163>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        template<> struct ConwayPolynomial<163, 19> { using ZPZ = aerobus::zpz<163>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
04818
                        ZPZV<0>, ZPZV<6</pre>
04819
                                       template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
                       POLYV<ZPZV<1>, ZPZV<162»; }; // NOLINT
04820
                                     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 =
 04821
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162»; }; // NOLINT template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; }; // NOLINT
 04823
                                    template<> struct ConwayPolynomial<167, 5> { using ZPZ = aerobus::zpz<167>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<162»; }; // NOLINT
                                      template<> struct ConwayPolynomial<167, 6> { using ZPZ = aerobus::zpz<167>; using type =
04824
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<75>, ZPZV<38>, ZPZV<2>, ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<167, 7> { using ZPZ = aerobus::zpz<167>; using type
04825
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162»; }; //
                                     template<> struct ConwayPolynomial<167, 8> { using ZPZ = aerobus::zpz<167>; using type =
 04826
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<149>, ZPZV<56>, ZPZV<113>, ZPZV<5»; }; //
                       NOLINT
04827
                                       template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<165>, ZPZV<162»;
                       }; // NOLINT
                                     template<> struct ConwayPolynomial<167, 10> { using ZPZ = aerobus::zpz<167>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<85>, ZPZV<68>, ZPZV<109>, ZPZV<143>,
                        ZPZV<148>, ZPZV<5»; }; // NOLINT
                                     template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type =
04829
                        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<
                       ZPZV<140>, ZPZV<41>, ZPZV<57>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 13> { using ZPZ = aerobus::zpz<167>; using type =
04831
                        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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04833
                        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 =
04835
                       POLYV<ZPZV<1>, ZPZV<169>, ZPZV<2»; }; // NOLINT
                                       template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
 04836
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
 04837
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<22; }; // NOLINT template<> struct ConwayPolynomial<173, 5> { using ZPZ = aerobus::zpz<173>; using type =
 04838
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<171»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<173, 6> { using ZPZ = aerobus::zpz<173>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<134>, ZPZV<107>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<173, 7> { using ZPZ = aerobus::zpz<173>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5171»; }; // NOLINT template<> struct ConwayPolynomial<173, 8> { using ZPZ = aerobus::zpz<173>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<125>, ZPZV<158>, ZPZV<27>, ZPZV<27>, ZPZV<2»; }; //
                          NOLINT
                                            template<> struct ConwayPolynomial<173, 9> { using ZPZ = aerobus::zpz<173>; using type
04842
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<104>, ZPZV<171»;
                          }; // NOLINT
04843
                                          template<> struct ConwayPolynomial<173, 10> { using ZPZ = aerobus::zpz<173>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<164>, ZPZV<48>, ZPZV<106>,
                                                                                                                                // NOLINT
                          ZPZV<58>, ZPZV<2»; };</pre>
                                            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 =
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                                            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»; };</pre>
                                                                                                                                                                                                                 // NOLINT
                                          template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type =
04847
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<171»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                                 // NOLINT
                                            template<> struct ConwayPolynomial<173, 19> { using ZPZ = aerobus::zpz<173>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                                                      ZPZV<0>,
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171»; }; //</pre>
                          NOLINT
04849
                                           template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<177»; }; // NOLINT
04850
                                            template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2»; }; // NOLINT
04851
                                         template<> struct ConwayPolynomial<179, 3> { using ZPZ = aerobus::zpz<179>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4+, ZPZV<177»; ); // NOLINT
template<> struct ConwayPolynomial<179, 4> { using ZPZ = aerobus::zpz<179>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<179, 5> { using ZPZ = aerobus::zpz<179>; using type =
04852
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<177»; }; // NOLINT
                                          template<> struct ConwayPolynomial<179, 6> { using ZPZ = aerobus::zpz<179>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<91>, ZPZV<55>, ZPZV<109>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
04855
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<177»; }; // NOLINT
                                           template<> struct ConwayPolynomial<179, 8> { using ZPZ = aerobus::zpz<179>; using type =
04856
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<63>, ZPZV<144>, ZPZV<73>, ZPZV<2»; };
                          NOLINT
04857
                                         template<> struct ConwayPolynomial<179, 9> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<64>, ZPZV<64>, ZPZV<177»; };
                           // NOLINT
                                           template<> struct ConwayPolynomial<179, 10> { using ZPZ = aerobus::zpz<179>; using type =
04858
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>>, ZPZV<71>, ZPZV<150>, ZPZV<49>, ZPZV<87>,
                          ZPZV<2»; }; // NOLINT</pre>
04859
                                         template<> struct ConwayPolynomial<179, 11> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04860
                                            template<> struct ConwayPolynomial<179, 12> { using ZPZ = aerobus::zpz<179>; using type
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<103>, ZPZV<83>, ZPZV<43>, ZPZV<76>, ZPZV<8>, ZPZV<177>, ZPZV<1>, ZPZV<2»; }; // NOLINT
                                            template<> struct ConwayPolynomial<179, 13> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<177»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<179, 17> { using ZPZ = aerobus::zpz<179>; using type =
04862
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<11>, ZPZV<177»; }; //</pre>
                          NOLINT
                                            template<> struct ConwayPolynomial<181, 1> { using ZPZ = aerobus::zpz<181>; using type =
                          POLYV<ZPZV<1>, ZPZV<179»; }; // NOLINT
                                            template<> struct ConwayPolynomial<181, 2> { using ZPZ = aerobus::zpz<181>; using type =
                          POLYV<ZPZV<1>, ZPZV<177>, ZPZV<2»; }; // NOLINT
                                            template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
04866
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179»; }; // NOLINT template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<105>, ZPZV<2»; }; // NOLINT
04867
                                          template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
04868
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<179»; }; // NOLINT
04869
                                           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 CommayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type = DNIV(ZPZV); ZPZV<0 | ZPZ
04870
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<179»; };
                                           template<> struct ConwayPolynomial<181, 8> { using ZPZ = aerobus::zpz<181>; using type
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<108>, ZPZV<22>, ZPZV<149>, ZPZV<2»; };
                          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>
04872
```

```
}; // NOLINT
04873
                                                   template<> struct ConwayPolynomial<181, 10> { using ZPZ = aerobus::zpz<181>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<154>, ZPZV<104>, ZPZV<94>, ZPZV<57>, ZPZV<88>,
                               ZPZV<2»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<181, 11> { using ZPZ = aerobus::zpz<181>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<24>, ZPZV<179»; }; // NOLINT</pre>
04874
                                                   template<> struct ConwayPolynomial<181, 12> { using ZPZ = aerobus::zpz<181>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<141>, ZPZV<45>, ZPZV<122>,
                              ZPZV<175>, ZPZV<12>, ZPZV<10>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<181, 13> { using ZPZ = aerobus::zpz<181>; using type =
04876
                               \texttt{POLYV} < \texttt{ZPZV} < \texttt{0} >, \ \texttt{ZPZV} < \texttt{
                              ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<179»; };</pre>
                                                                                                                                                                                                                                                    // NOLINT
                                                   template<> struct ConwayPolynomial<181, 17> { using ZPZ = aerobus::zpz<181>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0 , ZPZ
                                                   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 =
04880
                            POLYV<ZPZV<1>, ZPZV<190>, ZPZV<199»; }; // NOLINT template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
04881
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172»; }; // NOLINT
04882
                                                template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; }; // NOLINT template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
04883
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<17>; ; // NOLINT

template<> struct ConwayPolynomial<191, 6> { using ZPZ = aerobus::zpz<191>; using type = DIVV<ZPZVV1 = ZPZV<0 > ZPZV<1 > ZPZV<10 > Z
04884
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<110>, ZPZV<10>, ZPZV<19»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<191, 7> { using ZPZ = aerobus::zpz<191>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<142»; };
                              template<> struct ConwayPolynomial<191, 8> { using ZPZ = aerobus::zpz<191>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<164>, ZPZV<139>, ZPZV<171>, ZPZV<19»; }; //
                              NOLINT
                                                 template<> struct ConwayPolynomial<191, 9> { using ZPZ = aerobus::zpz<191>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<62>, ZPZV<124>, ZPZV<172»;
                               }; // NOLINT
04888
                                                   template<> struct ConwayPolynomial<191, 10> { using ZPZ = aerobus::zpz<191>; using type =
                               \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 113>, \ \texttt{ZPZV} < 47>, \ \texttt{ZPZV} < 173>, \ \texttt{ZPZV} < 74>, \ \texttt{ZPZV} < 7
                               ZPZV<156>, ZPZV<19»; }; // NOLINT</pre>
04889
                                                  template<> struct ConwayPolynomial<191, 11> { using ZPZ = aerobus::zpz<191>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                       // NOLINT
04890
                                               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</pre>
                                                  template<> struct ConwayPolynomial<191, 13> { using ZPZ = aerobus::zpz<191>; using type =
04891
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<172»; }; // NOLINT</pre>
04892
                                               template<> struct ConwayPolynomial<191,
                                                                                                                                                                                                                                                              17> { using ZPZ = aerobus::zpz<191>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              template<> struct ConwayPolynomial<191, 19> { using ZPZ = aerobus::zpz<191>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04893
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<190>, ZPZV<190>, ZPZV<2>, ZPZV<172»; }; //</pre>
04894
                                                  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 =
04895
                            POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
04897
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
04898
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<188»; }; // NOLINT
04899
                                                    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
04900
                                               template<> struct ConwayPolynomial<193, 7> { using ZPZ = aerobus::zpz<193>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 8> { using ZPZ = aerobus::zpz<193>; using type =
04901
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<145>, ZPZV<34>, ZPZV<154>, ZPZV<154>, ZPZV<5»; }; //
04902
                                               template<> struct ConwayPolynomial<193, 9> { using ZPZ = aerobus::zpz<193>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<168>, ZPZV<168>, ZPZV<27>, ZPZV<188»;
                               }; // NOLINT
04903
                                                 template<> struct ConwayPolynomial<193. 10> { using ZPZ = aerobus::zpz<193>: using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<51>, ZPZV<77>, ZPZV<6>, ZPZV<89>,
                               ZPZV<5»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type
                              POLYV<2PZV<1>, 2PZV<0>, 2PZV<0
                               ZPZV<1>, ZPZV<188»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<193, 12> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<5>, ZPZV<52>, ZPZV<135>, ZPZV<152>,
04905
```

```
ZPZV<90>, ZPZV<46>, ZPZV<28>, ZPZV<5»; };</pre>
                                                                                                                                                                                                         // NOLINT
                                          template<> struct ConwayPolynomial<193, 13> { using ZPZ = aerobus::zpz<193>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<3>, ZPZV<188»; }; // NOLINT
   template<> struct ConwayPolynomial<193, 17> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04907
                          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
 04908
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<188»; }; //</pre>
                          NOLINT
                                        template<> struct ConwayPolynomial<197, 1> { using ZPZ = aerobus::zpz<197>; using type =
04909
                         POLYV<ZPZV<1>, ZPZV<195»; }; // NOLINT
                                           template<> struct ConwayPolynomial<197, 2> { using ZPZ = aerobus::zpz<197>; using type =
                          POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2»; }; // NOLINT
 04911
                                          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 =
04912
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<124>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<197, 5> { using ZPZ = aerobus::zpz<197>; using type =
 04913
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195»; }; // NOLINT
 04914
                                          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 =
04915
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<195»; };
                                          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
04917
                                          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<0>, ZPZV<13>, ZPZV<127>, ZPZV<8>, ZPZV<195»;
                          }; // NOLINT
04918
                                           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>
                         \label{eq:convayPolynomial} $$$ template<> struct ConwayPolynomial<197, 11> { using ZPZ = aerobus::zpz<197>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04919
                          ZPZV<14>, ZPZV<195»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<197, 12> { using ZPZ = aerobus::zpz<197>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<168>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>,
                          ZPZV<90>, ZPZV<163>, ZPZV<2»; }; // NOLINT</pre>
04921
                                        template<> struct ConwayPolynomial<197, 13> { using ZPZ = aerobus::zpz<197>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          template<> struct ConwayPolynomial<197, 17> { using ZPZ = aerobus::zpz<197>; using type =
04922
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<195»; }; // NOLINT
template<> struct ConwayPolynomial<197, 19> { using ZPZ = aerobus::zpz<197>; using type =
04923
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04924
                                          template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
                          POLYV<ZPZV<1>, ZPZV<196»; }; // NOLINT
 04925
                                          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 =
04926
                        POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<196»; }; // NOLINT template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; }; // NOLINT
                                           template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
 04928
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
04929
                                        template<> struct ConwayPolynomial<199, 6> { using ZPZ = aerobus::zpz<199>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<90>, ZPZV<58>, ZPZV<79>, ZPZV<3»; }; // NOLINT
 04930
                                          template<> struct ConwayPolynomial<199, 7> { using ZPZ = aerobus::zpz<199>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<196»; }; //
04931
                                        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 =
04932
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<141>, ZPZV<196»;
                          }; // NOLINT
                                          template<> struct ConwayPolynomial<199, 10> { using ZPZ = aerobus::zpz<199>; using type =
04933
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<158>, ZPZV<31>, ZPZV<54>, ZPZV<9>,
                          ZPZV<3»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<199, 11> { using ZPZ = aerobus::zpz<199>; using type =
04934
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; }; // NOLINT
                                        template<> struct ConwayPolynomial<199, 12> { using ZPZ = aerobus::zpz<199>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<33>, ZPZV<192>, ZPZV<197>, ZPZV<138>,
                         ZPZV<69>, ZPZV<57>, ZPZV<151>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<199, 13> { using ZPZ = aerobus::zpz<199>; using type =
04936
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<196»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<199, 17> { using ZPZ = aerobus::zpz<199>; using type
                         POLYV<2PZV<1>, 2PZV<0>, 2PZV<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>,
 04938
```

```
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<196; }; //</pre>
04939
                                    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 =
04940
                       POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; }; // NOLINT
                                       template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<209»; }; // NOLINT
                                     template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
 04942
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<211, 5> { using ZPZ = aerobus::zpz<211>; using type =
04943
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<209»; }; // NOLINT
04944
                                      template<> struct ConwayPolynomial<211, 6> { using ZPZ = aerobus::zpz<211>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<81>, ZPZV<194>, ZPZV<133>, ZPZV<2»; }; // NOLINJ
 04945
                                    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<0>, ZPZV<3>, ZPZV<3>, ZPZV<30, ZPZV<209»; }; // NOLINT template<> struct ConwayPolynomial<211, 8> { using ZPZ = aerobus::zpz<211>; using type =
04946
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<200>, ZPZV<87>, ZPZV<29>, ZPZV<29; };
                                      template<> struct ConwayPolynomial<211, 9> { using ZPZ = aerobus::zpz<211>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<139>, ZPZV<139>, ZPZV<26>, ZPZV<209»;
                       }; // NOLINT
04948
                                      template<> struct ConwayPolynomial<211, 10> { using ZPZ = aerobus::zpz<211>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<30>, ZPZV<61>, ZPZV<148>, ZPZV<148>, ZPZV<87>, ZPZV<125>,
                       ZPZV<2»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<211, 11> { using ZPZ = aerobus::zpz<211>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<7>, ZPZV<209»; }; // NOLINT</pre>
04950
                                      template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<50>, ZPZV<145>, ZPZV<126>, ZPZV<184>, ZPZV<84>, ZPZV<27>, ZPZV<284>, 
04951
                                       template<> struct ConwayPolynomial<211,
                                                                                                                                                                                             13> { using ZPZ = aerobus::zpz<211>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       template<> struct ConwayPolynomial<211, 17> { using ZPZ = aerobus::zpz<211>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04952
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<209»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<211, 19> { using ZPZ = aerobus::zpz<211>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<209»; }; //</pre>
                       NOLINT
04954
                                      template<> struct ConwayPolynomial<223, 1> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYV<ZPZV<1>, ZPZV<220»; }; // NOLINT
                                      template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
04955
                       POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; }; // NOLINT
 04956
                                      template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<20>»; }; // NOLINT
template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<223, 5> { using ZPZ = aerobus::zpz<223>; using type =
04957
 04958
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<20»; }; // NOLINT
                                      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
04960
                                     template<> struct ConwayPolynomial<223, 7> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<20»; }; // NOLINT
                                      template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
04961
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<3»; }; //
                       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
                       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>,
04963
                       ZPZV<3»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<223, 11> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<8>, ZPZV<220»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<223, 12> { using ZPZ = aerobus::zpz<223>; using type =
04965
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<64>, ZPZV<94>, ZPZV<11>, ZPZV<105>, ZPZV<64>,
                       ZPZV<151>, ZPZV<213>, ZPZV<3»; }; // NOLINT</pre>
                                      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
04967
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; };</pre>
                                    template<> struct ConwayPolynomial<223, 19> { using ZPZ = aerobus::zpz<223>; using type
 04968
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<220»; }; //</pre>
                       NOLINT
04969
                                      template<> struct ConwayPolynomial<227, 1> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<225»; }; // NOLINT
                                      template<> struct ConwayPolynomial<227, 2> { using ZPZ = aerobus::zpz<227>; using type =
                     POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2»; }; // NOLINT
 04971
                                    template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<225»; }; // NOLINT template<> struct ConwayPolynomial<227, 4> { using ZPZ = aerobus::zpz<227>; using type =
 04972
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2»; };
                                      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
                                    template<> struct ConwayPolynomial<227, 6> { using ZPZ = aerobus::zpz<227>; using type =
 04974
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<24>, ZPZV<135>, ZPZV<2»; }; // NOLINT
04975
                                     template<> struct ConwayPolynomial<227,
                                                                                                                                                                                          7> { using ZPZ = aerobus::zpz<227>; using type
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<225»; }; // NOLINT
 04976
                                      template<> struct ConwayPolynomial<227, 8> { using ZPZ = aerobus::zpz<227>;
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<151>, ZPZV<176>, ZPZV<106>, ZPZV<2»; }; //
                       NOLINT
04977
                                    template<> struct ConwayPolynomial<227, 9> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<28»;
                      }; // NOLINT template<> struct ConwayPolynomial<227, 10> { using ZPZ = aerobus::zpz<227>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>, ZPZV<12>, ZPZV<12>, ZPZV<93>, ZPZV<77>,
                       ZPZV<2»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<123>, ZPZV<99>, ZPZV<160>, ZPZV<96>, ZPZV<127>, ZPZV<142>, ZPZV<94>, ZPZV<22*; }; // NOLINT template<> struct ConwayPolynomial<227, 13> { using ZPZ = aerobus::zpz<227>; using type =
04981
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<225»; };</pre>
                                                                                                                                                                                  // NOLINT
                                      template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                       // NOLINT
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<225»; };</pre>
04983
                                      template<> struct ConwayPolynomial<227, 19> { using ZPZ = aerobus::zpz<227>; using type =
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<225»; }; //</pre>
                       NOLINT
                                      template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
 04984
                      POLYV<ZPZV<1>, ZPZV<223»; }; // NOLINT
 04985
                                      template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»; }; // NOLINT
                                     template<> struct ConwayPolynomial229, 3> { using ZPZ = aerobus::zpz<229>; using type =
04986
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; }; // NOLINT
 04987
                                      template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<7>, ZPZV<162>, ZPZY<6»; }; // NOLINT template<> struct ConwayPolynomial<229, 5> { using ZPZ = aerobus::zpz<229>; using type =
 04988
                     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 =
 04989
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<160>, ZPZV<186>, ZPZV<6>; }; // NOLINT
                                      template<> struct ConwayPolynomial<229, 7> { using ZPZ = aerobus::zpz<229>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<23»; };
 04991
                                    template<> struct ConwayPolynomial<229, 8> { using ZPZ = aerobus::zpz<229>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<193>, ZPZV<62>, ZPZV<205>, ZPZV<6»; }; //
                      NOLINT
                                     template<> struct ConwayPolynomial<229, 9> { using ZPZ = aerobus::zpz<229>; using type =
04992
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<117>, ZPZV<50>, ZPZV<223»;
                       }; // NOLINT
04993
                                      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 =
04994
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<2>, ZPZV<223»; };</pre>
                                                                                                                // NOLINT
                      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<145>, ZPZV<6»; }; // NOLINT
                                    template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type =
04996
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<223»; };</pre>
                                                                                                                                                                                   // NOLINT
                                    template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type =
04997
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<22 ; }; // NOLINT
template<> struct ConwayPolynomial<229, 19> { using ZPZ = aerobus::zpz<229>; using type =
04998
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<223»; }; //</pre>
                                    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 =
05000
                      POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
                                      template<> struct ConwayPolynomial<233, 3> { using ZPZ = aerobus::zpz<233>; using type =
                      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
 05002
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<3»; }; // NOLINT
                                     template<> struct ConwayPolynomial<233, 5> { using ZPZ = aerobus::zpz<233>; using type =
 05003
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<230»; }; // NOLINT
 05004
                                      template<> struct ConwayPolynomial<233, 6> { using ZPZ = aerobus::zpz<233>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<122>, ZPZV<215>, ZPZV<3>, ZPZV<32>, ZPZV<32>, ZPZV<32>, ZPZV<32>, ZPZV<32>; ); // NOLINT template<> struct ConwayPolynomial<233, 7> { using ZPZ = aerobus::zpz<233>; using type =
 05005
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 8> { using ZPZ = aerobus::zpz<233>; using type =
 05006
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<202>, ZPZV<135>, ZPZV<181>, ZPZV<3»; }; //
```

```
NOLINT
05007
                                        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
05008
                                          template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type :
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<28>, ZPZV<71>, ZPZV<102>, ZPZV<3>, ZPZV<48>,
                         ZPZV<3»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<233, 11> { using ZPZ = aerobus::zpz<233>; using type
05009
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        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>,
05010
                         ZPZV<216>, ZPZV<20>, ZPZV<3»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<233, 13> { using ZPZ = aerobus::zpz<233>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         \mbox{ZPZV}<0> , \mbox{ZPZV}<0> , \mbox{ZPZV}<14> , \mbox{ZPZV}<230 »; }; // NOLINT
                                         template<> struct ConwayPolynomial<233, 17> { using ZPZ = aerobus::zpz<233>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43, ZPZV<230»; }; // NOLINT</pre>
                                        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<25>, ZPZV<230»; }; //</pre>
                         NOLINT
                                        template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
05014
                        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
05016
                                        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 =
05017
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<239, 5> { using ZPZ = aerobus::zpz<239>; using type =
05018
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; }; // NOLINT
05019
                                      template<> struct ConwayPolynomial<239, 6> { using ZPZ = aerobus::zpz<239>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<237>, ZPZV<60>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<239, 7> { using ZPZ = aerobus::zpz<239>; using type =
05020
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<232»; }; // NOLINT
                                        template<> struct ConwayPolynomial<239, 8> { using ZPZ = aerobus::zpz<239>; using type =
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<202>, ZPZV<54>, ZPZV<7»; };
05022
                                      template<> struct ConwayPolynomial<239, 9> { using ZPZ = aerobus::zpz<239>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<88>, ZPZV<232»; };
                         // NOLINT
05023
                                         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>,
                                                                                                                          // NOLINT
                          ZPZV<108>, ZPZV<7»; };</pre>
05024
                                       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 =
05025
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<235>, ZPZV<14>, ZPZV<113>, ZPZV<182>, ZPZV<101>, ZPZV<81>, ZPZV<216>, ZPZV<7»; }; // NOLINT
05026
                                      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 =
05027
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<232»; };</pre>
                                      template<> struct ConwayPolynomial<239, 19> { using ZPZ = aerobus::zpz<239>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2
                         NOLINT
05029
                                         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
05031
                                         template<> struct ConwayPolynomial<241, 3> { using ZPZ = aerobus::zpz<241>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<234»; }; // NOLINT
template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<152>, ZPZV<7»; }; // NOLINT
                                        template<> struct ConwayPolynomial<241, 5> { using ZPZ = aerobus::zpz<241>; using type =
05033
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<234»; }; // NOLINT
                       template<> struct ConwayPolynomial<241, 6> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<6>, ZPZV<5>, ZPZV<7»; }; // NOLINT</pre>
05034
                        template<> struct ConwayPolynomial<241, 7> { using ZPZ = aerobus::zpz<241>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<234»; }; // NOLINT
05035
                                      template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<173>, ZPZV<212>, ZPZV<153>, ZPZV<7»; }; //
                         NOLINT
05037
                                        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<236>, ZPZV<236>, ZPZV<236»;
                        }; // NOLINT
  template<> struct ConwayPolynomial<241, 10> { using ZPZ = aerobus::zpz<241>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<27>, ZPZV<145>, ZPZV<208>, ZPZV<55>,
                        ZPZV<7»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05039
```

```
ZPZV<3>, ZPZV<234»; };</pre>
                                           template<> struct ConwayPolynomial<241, 12> { using ZPZ = aerobus::zpz<241>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<10>, ZPZV<109>, ZPZV<168>, ZPZV<22>,
                         ZPZV<197>, ZPZV<17>, ZPZV<7»; }; // NOLINT
  template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type :
0.5041
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           template<> struct ConwayPolynomial<241,
                                                                                                                                                                                                                         17> { using ZPZ = aerobus::zpz<241>; using type
05042
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<234»; }; // NOLINT
    template<> struct ConwayPolynomial<241, 19> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
05043
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<234w; }; //</pre>
05044
                                         template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
                         POLYV<ZPZV<1>, ZPZV<245»; }; // NOLINT template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
05045
                         POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; }; // NOLINT
                                           template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<200>, ZPZV<6»; }; // NOLINT
05048
                                         template<> struct ConwayPolynomial<251, 5> { using ZPZ = aerobus::zpz<251>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT
05049
                                           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 =
05050
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<245»; }; // NOLINT template<> struct ConwayPolynomial<251, 8> { using ZPZ = aerobus::zpz<251>; using type =
05051
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<142>, ZPZV<215>, ZPZV<173>, ZPZV<6»; }; //
                         NOLINT
05052
                                           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
                         template<> struct ConwayPolynomial<251, 10> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<138>, ZPZV<110>, ZPZV<45>, ZPZV<34>,
05053
                                                                                                                                  // NOLINT
                          ZPZV<149>, ZPZV<6»; };</pre>
                                           template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<26>, ZPZV<245»; }; // NOLINT</pre>
05055
                                         template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<192>, ZPZV<53>, ZPZV<20>, ZPZV<20>, ZPZV<15>,
                          ZPZV<201>, ZPZV<232>, ZPZV<6»; }; // NOLINT</pre>
05056
                                           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>
05057
                                         template<> struct ConwayPolynomial<251, 17> { using ZPZ = aerobus::zpz<251>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>; ZPZV<0
05058
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<84>, ZPZV<245»; }; //</pre>
                          NOLINT
05059
                                           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 =
05060
                          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 =
05062
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<18>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<257, 5> { using ZPZ = aerobus::zpz<257>; using type =
05063
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<254»; }; // NOLINT
                                           template<> struct ConwayPolynomial<257, 6> { using ZPZ = aerobus::zpz<257>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<62>, ZPZV<18>, ZPZV<138>, ZPZV<3»; }; // NOLINT
                                         template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type =
05065
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<254»; }; // NOLINT template<> struct ConwayPolynomial<257, 8> { using ZPZ = aerobus::zpz<257>; using type =
05066
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<179>, ZPZV<140>, ZPZV<162>, ZPZV<3»; }; //
                         NOLINT
                                           template<> struct ConwayPolynomial<257, 9> { using ZPZ = aerobus::zpz<257>; using type =
05067
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<201>, ZPZV<50>, ZPZV<
                          }; // NOLINT
                                           template<> struct ConwayPolynomial<257, 10> { using ZPZ = aerobus::zpz<257>; using type =
05068
                          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
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<40>, ZPZV<254»; };
                                                                                                                                 // NOLINT
05070
                                         template<> struct ConwayPolynomial<257, 12> { using ZPZ = aerobus::zpz<257>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<25>, ZPZV<215>, 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
                         ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<254»; }; // NOLINT
   template<> struct ConwayPolynomial<257, 17> { using ZPZ = aerobus::zpz<257>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
05072
```

```
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<ZPZV<1>, ZPZV<0>, ZPZV<0
              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<254»; }; //</pre>
              NOLINT
05074
                        template<> struct ConwavPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<ZPZV<1>, ZPZV<258»; }; // NOLINT
05075
                         template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; }; // NOLINT
05076
                       template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; }; // NOLINT
template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
05077
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; };
                                                                                                                                                         // NOLINT
                         template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<258»; }; // NOLINT
05079
                        template<> struct ConwayPolynomial<263, 6> { using ZPZ = aerobus::zpz<263>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<22>, ZPZV<250>, ZPZV<225>, ZPZV<255>, ZPZV<5>; }; // NOLINT template<> struct ConwayPolynomial<263, 7> { using ZPZ = aerobus::zpz<263>; using type
05080
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<258»; }; // NOLINT
                        template<> struct ConwayPolynomial<263, 8> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<227>, ZPZV<170>, ZPZV<7>, ZPZV<5»; };
              NOLINT
05082
                       template<> struct ConwayPolynomial<263, 9> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<261>, ZPZV<29>, ZPZV<258»;
              }; // NOLINT
                         template<> struct ConwayPolynomial<263, 10> { using ZPZ = aerobus::zpz<263>; using type
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<245>, ZPZV<231>, ZPZV<198>, ZPZV<145>,
               ZPZV<119>, ZPZV<5»; }; // NOLINT</pre>
05084
                        template<> struct ConwayPolynomial<263, 11> { using ZPZ = aerobus::zpz<263>; using type :
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<258»; }; // NOLINT
              template<> struct ConwayPolynomial<263, 12> { using ZPZ = aerobus::zpz<263>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<174>, ZPZV<174>, ZPZV<162>, ZPZV<252>,
05085
              05086
                        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 =
05087
              POLYV<ZPZV<1>, ZPZV<268>, ZPZV<2»; }; // NOLINT
05088
                         template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
              POLYY<ZPZY<1>, ZPZY<0>, ZPZY<9>, ZPZY<267»; }; // NOLINT template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
05089
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<269, 5> { using ZPZ = aerobus::zpz<269>; using type =
05090
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<267»; }; // NOLINT
                         template<> struct ConwayPolynomial<269, 6> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<120>, ZPZV<101>, ZPZV<206>, ZPZV<2»; };
05092
                       template<> struct ConwayPolynomial<269, 7> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<66>, ZPZV<267»; }; // NOLINT
05093
                       template<> struct ConwayPolynomial<269, 8> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<220>, ZPZV<131>, ZPZV<232>, ZPZV<23; }; //
                        template<> struct ConwayPolynomial<269, 9> { using ZPZ = aerobus::zpz<269>; using type =
05094
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<214>, ZPZV<267>, ZPZV<267>,
              }; // NOLINT
                        template<> struct ConwayPolynomial<269, 10> { using ZPZ = aerobus::zpz<269>; using type =
05095
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<264>, ZPZV<243>, ZPZV<186>, ZPZV<61>, ZPZV<20>, ZP
                       template<> struct ConwayPolynomial<269, 11> { using ZPZ = aerobus::zpz<269>; using type
05096
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
              ZPZV<20>, ZPZV<267»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type =
05097
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<126>, ZPZV<165>, ZPZV<165>, ZPZV<63>, ZPZV<215>, ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2»; }; // NOLINT
                         template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<265»; }; // NOLINT
05099
                       template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; }; // NOLINT
                        template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
05100
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT
                         template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type =
05101
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6»; }; // NOLINT
                       template<> struct ConwayPolynomial<271, 5> { using ZPZ = aerobus::zpz<271>; using type =
05102
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT template<> struct ConwayPolynomial<271, 6> { using ZPZ = aerobus::zpz<271>; using type =
05103
              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<207>, ZPZV<207>, ZPZV<81>, ZPZV<6;; ; // NOLINT template<> struct ConwayPolynomial<271, 7> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; };
05105
                       template<> struct ConwayPolynomial<271, 8> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<114>, ZPZV<69>, ZPZV<69>, ZPZV<60>, ZPZV<10>, ZPZV<10
              NOLINT
                        template<> struct ConwayPolynomial<271, 9> { using ZPZ = aerobus::zpz<271>; using type =
05106
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<26>, ZPZV<26 , ZPZV<2
                        template<> struct ConwayPolynomial<271, 10> { using ZPZ = aerobus::zpz<271>; using type =
05107
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<133>, ZPZV<10>, ZPZV<256>, ZPZV<74>, ZPZV<126>, ZPZV<6»; }; // NOLINT
05108
                      template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
              ZPZV<10>, ZPZV<265»; }; // NOLINT</pre>
05109
                       template<> struct ConwayPolynomial<271, 12> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162>, ZPZV<210>, ZPZV<216>, ZPZV<215>,
              ZPZV<237>, ZPZV<256>, ZPZV<130>, ZPZV<6»; }; // NOLINT
template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
05110
              POLYV<ZPZV<1>, ZPZV<272»; }; // NOLINT
                        template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
05111
              POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; }; // NOLINT
                      template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
05112
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; }; // NOLINT template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
05113
             POLYV<ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<277, 5> { using ZPZ = aerobus::zpz<277>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<272»; }; // NOLINT
05115
                      template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<9>, ZPZV<118>, ZPZV<5»; }; // NOLINT
                       template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type
05116
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<272»; }; // NOLINT
                       template<> struct ConwayPolynomial<277, 8> { using ZPZ = aerobus::zpz<277>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<159>, ZPZV<176>, ZPZV<5»; }; //
              NOLINT
0.5118
                      template<> struct ConwayPolynomial<277, 9> { using ZPZ = aerobus::zpz<277>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177>, ZPZV<110>, ZPZV<272»;
              }; // NOLINT
                        template<> struct ConwayPolynomial<277, 10> { using ZPZ = aerobus::zpz<277>; using type
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<206>, ZPZV<253>, ZPZV<237>, ZPZV<241>,
              ZPZV<260>, ZPZV<5»; }; // NOLINT</pre>
05120
                        template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type :
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05121
                        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>
05122
                       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 =
05123
              POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; }; // NOLINT
05124
                        template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
              POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<278»; }; // NOLINT template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
05125
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<281, 5> { using ZPZ = aerobus::zpz<281>; using type =
05126
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<278»; }; // NOLINT
                        template<> struct ConwayPolynomial<281, 6> { using ZPZ = aerobus::zpz<281>; using type =
                                                                                                                                                                                                    // NOLINT
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<151>, ZPZV<13>, ZPZV<27>, ZPZV<3»; };
05128
                      template<> struct ConwayPolynomial<281, 7> { using ZPZ = aerobus::zpz<281>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<278»; }; // NOLINT
                      template<> struct ConwayPolynomial<281, 8> { using ZPZ = aerobus::zpz<281>; using type =
05129
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<49>, ZPZV<279>, ZPZV<140>, ZPZV<3»; }; //
                       template<> struct ConwayPolynomial<281, 9> { using ZPZ = aerobus::zpz<281>; using type =
05130
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<70>, ZPZV<70>,
              }; // NOLINT
05131
                       template<> struct ConwayPolynomial<281, 10> { using ZPZ = aerobus::zpz<281>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<145>, ZPZV<13>, ZPZV<138>, ZPZV<191>, ZPZV<3»; }; // NOLINT
                      template<> struct ConwayPolynomial<281, 11> { using ZPZ = aerobus::zpz<281>; using type
05132
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
              ZPZV<36>, ZPZV<278»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type =
05133
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<202>, ZPZV<68>, ZPZV<103>, ZPZV<116>,
              ZPZV<58>, ZPZV<28>, ZPZV<191>, ZPZV<3»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
              POLYV<ZPZV<1>, ZPZV<280»; }; // NOLINT
05135
                       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 =
05136
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
                        template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type =
05137
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; }; // NOLINT
                      template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
05138
             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<38»; }; // NOLINT template<> struct ConwayPolynomial<283, 6> { using ZPZ = aerobus::zpz<283>; using type =
05139
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<68>, ZPZV<73>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<280»; };
05141
                      template<> struct ConwayPolynomial<283, 8> { using ZPZ = aerobus::zpz<283>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<179>, ZPZV<32>, ZPZV<232>, ZPZV<3»; }; //
              NOLINT
                       template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type =
05142
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      template<> struct ConwayPolynomial<283, 10> { using ZPZ = aerobus::zpz<283>; using type :
05143
              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<271>, ZPZV<185>, ZPZV<68>, ZPZV<100>, ZPZV<219>, ZPZV<3»; }; // NOLINT
05144
                      template<> struct ConwayPolynomial<283, 11> { using ZPZ = aerobus::zpz<283>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
         ZPZV<4>, ZPZV<280»; }; // NOLINT</pre>
05145
              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<20>, ZPZV<8>, ZPZV<96>, ZPZV<229>, ZPZV<49>, ZPZV<14>, ZPZV<56>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<293, 1> { using ZPZ = aerobus::zpz<293>; using type =
05146
        POLYV<ZPZV<1>, ZPZV<291»; }; // NOLINT
05147
               template<> struct ConwayPolynomial<293, 2> { using ZPZ = aerobus::zpz<293>; using type =
        POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; }; // NOLINT
05148
              template<> struct ConwayPolynomial<293, 3> { using ZPZ = aerobus::zpz<293>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT template<> struct ConwayPolynomial<293, 4> { using ZPZ = aerobus::zpz<293>; using type =
05149
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
05151
              template<> struct ConwayPolynomial<293, 6> { using ZPZ = aerobus::zpz<293>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<210>, ZPZV<260>, ZPZV<2w; }; // NOLINT template<> struct ConwayPolynomial<293, 7> { using ZPZ = aerobus::zpz<293>; using type
05152
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<291»; }; // NOLINT
               template<> struct ConwayPolynomial<293, 8> { using ZPZ = aerobus::zpz<293>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<2»; }; //
         NOLINT
05154
              template<> struct ConwayPolynomial<293, 9> { using ZPZ = aerobus::zpz<293>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<291»;
         }; // NOLINT
               template<> struct ConwayPolynomial<293, 10> { using ZPZ = aerobus::zpz<293>; using type :
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<186>, ZPZV<28>, ZPZV<46>, ZPZV<184>, ZPZV<24>,
         ZPZV<2»; }; // NOLINT</pre>
        template<> struct ConwayPolynomial<293, 11> { using ZPZ = aerobus::zpz<293>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<291»; }; // NOLINT</pre>
05156
05157
               template<> struct ConwayPolynomial<293, 12> { using ZPZ = aerobus::zpz<293>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<159>, ZPZV<210>, ZPZV<125>, ZPZV<212>,
         ZPZV<167>, ZPZV<144>, ZPZV<157>, ZPZV<2»; }; // NOLINT</pre>
05158
               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 =
05159
        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 =
05161
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<239>, ZPZV<239>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<307, 5> { using ZPZ = aerobus::zpz<307>; using type =
0.5162
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<302»; }; // NOLINT
               template<> struct ConwayPolynomial<307, 6> { using ZPZ = aerobus::zpz<307>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<213>, ZPZV<172>, ZPZV<61>, ZPZV<5»; }; // NOLINT
05164
              template<> struct ConwayPolynomial<307, 7> { using ZPZ = aerobus::zpz<307>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<302»; }; // NOLINT
              template<> struct ConwayPolynomial<307, 8> { using ZPZ = aerobus::zpz<307>; using type =
05165
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<283>, ZPZV<232>, ZPZV<131>, ZPZV<5»; }; //
               template<> struct ConwayPolynomial<307, 9> { using ZPZ = aerobus::zpz<307>; using type =
05166
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<165>, ZPZV<165>, ZPZV<70>, ZPZV<302»;
         }; // NOLINT
05167
               template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
        POLYV<ZPZV<1>, ZPZV<294»; }; // NOLINT
               template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
        POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; }; // NOLINT
               template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
05169
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; ); // NOLINT
template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; }; // NOLINT
template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
05170
05171
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294»; }; // NOLINT
05172
              template<> struct ConwayPolynomial<311, 6> { using ZPZ = aerobus::zpz<311>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<167>, ZPZV<152>, ZPZV<17»; }; // NOLINT
05173
              template<> struct ConwayPolynomial<311, 7> { using ZPZ = aerobus::zpz<311>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2, ZPZV<0>, ZPZV<2, ZPZV<10>, ZPZV<294»; }; // NOLINT template<> struct ConwayPolynomial<311, 8> { using ZPZ = aerobus::zpz<311>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<162>, ZPZV<118>, ZPZV<2>, ZPZV<17»; }; //
05175
              template<> struct ConwayPolynomial<311, 9> { using ZPZ = aerobus::zpz<311>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<287>, ZPZV<287>, ZPZV<74>, ZPZV<294»;
         }; // NOLINT
05176
               template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<ZPZV<1>, ZPZV<303»; }; // NOLINT
               template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
05177
        POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; }; // NOLINT
              template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
05178
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; }; // NOLINT
               template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
05179
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<239>, ZPZV<10»; };
                                                                                               // NOLINT
               template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<303»; }; // NOLINT
05181
              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 =
05182
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<3033»; };
                      template<> struct ConwayPolynomial<313, 8> { using ZPZ = aerobus::zpz<313>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<306>, ZPZV<99>, ZPZV<106>, ZPZV<10»; }; //
             NOLINT
0.5184
                      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<303»;
05185
                      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 =
05186
             POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; }; // NOLINT
                      template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
05187
            POLYV<ZPZV<1>, ZPZV<7>, ZPZV<7>, ZPZV<315»; }; // NOLINT template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2»; }; // NOLINT
05189
                      template<> struct ConwayPolynomial<317, 5> { using ZPZ = aerobus::zpz<317>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<315»; }; // NOLINT
             template<> struct ConwayPolynomial<317, 6> { using ZPZ = aerobus::zpz<317>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<195>, ZPZV<4>, ZPZV<2»; }; // NOLINT
05190
                      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
05192
                      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
05193
                      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
05194
                     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 =
05195
             POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; }; // NOLINT
05196
                      template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
             POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<328*; }; // NOLINT template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
05197
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<290>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<331, 5> { using ZPZ = aerobus::zpz<331>; using type =
05198
             POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<4>, ZPZY<328»; }; // NOLINT template<> struct ConwayPolynomial<331, 6> { using ZPZ = aerobus::zpz<331>; using type =
             POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<205>, ZPZV<159>, ZPZV<3»; }; // NOLINT
                     template<> struct ConwayPolynomial<331, 7> { using ZPZ = aerobus::zpz<331>; using type =
05200
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
05201
                     template<> struct ConwayPolynomial<331, 8> { using ZPZ = aerobus::zpz<331>; using type =
             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<249>, ZPZV<308>, ZPZV<78>, ZPZV<3»; };
                      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<1>, ZPZV<194>, ZPZV<210>, ZPZV<328»;
             }; // NOLINT
05203
                      template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
             POLYV<ZPZV<1>, ZPZV<327»; }; // NOLINT
                      template<> struct ConwayPolynomial<337, 2> { using ZPZ = aerobus::zpz<337>; using type =
05204
             POLYV<ZPZV<1>, ZPZV<332>, ZPZV<10»; }; // NOLINT
                      template<> struct ConwayPolynomial<337, 3> { using ZPZ = aerobus::zpz<337>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
05206
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<337, 5> { using ZPZ = aerobus::zpz<337>; using type =
05207
             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 =
05208
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<216>, ZPZV<109, ZPZV<109; }; // NOLINT template<> struct ConwayPolynomial<337, 7> { using ZPZ = aerobus::zpz<337>; using type
05209
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<327»; }; // NOLINT template<> struct ConwayPolynomial<337, 8> { using ZPZ = aerobus::zpz<337>; using type =
05210
             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 =
05211
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<148>, ZPZV<98>, ZPZV<327»;
             }; // NOLINT
                      \texttt{template<> struct ConwayPolynomial<347, 1> \{ using ZPZ = aerobus:: zpz<347>; using type = 200 t
05212
             POLYV<ZPZV<1>, ZPZV<345»; }; // NOLINT
05213
                      template<> struct ConwayPolynomial<347, 2> { using ZPZ = aerobus::zpz<347>; using type =
             POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; }; // NOLINT
05214
                     template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<345»; ); // NOLINT
template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
05215
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<345»; }; // NOLINT
05217
                      template<> struct ConwayPolynomial<347, 6> { using ZPZ = aerobus::zpz<347>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<343>, ZPZV<26>, ZPZV<56>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
05218
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<345»; }; // NOLINT
                      template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<187>, ZPZV<213>, ZPZV<117>, ZPZV<2»; };
             NOLINT
05220
                     \texttt{template<>} \texttt{struct ConwayPolynomial<347, 9> \{ \texttt{using ZPZ = aerobus::zpz<347>; using type = 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 2000 + 20
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<235>, ZPZV<252>, ZPZV<252>, ZPZV<345»;
             }; // NOLINT
```

```
template<> struct ConwayPolynomial<349, 1> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<347»; }; // NOLINT
           template<> struct ConwayPolynomial<349, 2> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; }; // NOLINT
05223
           template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; }; // NOLINT template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<349, 5> { using ZPZ = aerobus::zpz<349>; using type =
05225
      template<> struct ConwayPolynomial<349, 6> { using ZPZ = aerobus::zpz<349>; using type =
05226
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<135>, ZPZV<316>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<349, 7> { using ZPZ = aerobus::zpz<349>; using type
05227
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347»; }; //
05228
          template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<328>, ZPZV<268>, ZPZV<268; }; //
      NOLTNT
      template<> struct ConwayPolynomial<349, 9> { using ZPZ = aerobus::zpz<349>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<290>, ZPZV<130>, ZPZV<347»;
05229
           template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<350»; }; // NOLINT
           template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
05231
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
05232
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type =
05233
      05234
           template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2, ZPZV<1>, ZPZV<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 6> { using ZPZ = aerobus::zpz<353>; using type =
05235
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<215>, ZPZV<226>, ZPZV<295>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<353, 7> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<350»; };
           template<> struct ConwayPolynomial<353, 8> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<16>, ZPZV<26>, ZPZV<37>, ZPZV<3»; };
           template<> struct ConwayPolynomial<353, 9> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<319>, ZPZV<49>, ZPZV<350»;
      }; // NOLINT
05239
           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 =
05240
      POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<352»; }; // NOLINT
          template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
05242
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<229>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<359, 5> { using ZPZ = aerobus::zpz<359>; using type =
05243
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; }; // NOLINT
05244
           template<> struct ConwayPolynomial<359, 6> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<309>, ZPZV<327>, ZPZV<327>, ZPZV<7»; }; // NOLIN
05245
           template<> struct ConwayPolynomial<359, 7> { using ZPZ = aerobus::zpz<359>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<352»; }; // NOLINT template<> struct ConwayPolynomial<359, 8> { using ZPZ = aerobus::zpz<359>; using type =
05246
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30+, ZPZV<301>, ZPZV<143>, ZPZV<271>, ZPZV<27»; }; //
05247
           template<> struct ConwayPolynomial<359, 9> { using ZPZ = aerobus::zpz<359>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<356>, ZPZV<165>, ZPZV<352»;
      }; // NOLINT
05248
           template<> struct ConwayPolynomial<367, 1> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<361»; }; // NOLINT
           template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6»; }; // NOLINT
05250
           template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<36|»; }; // NOLINT
template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6»; }; // NOLINT
template<> struct ConwayPolynomial<367, 5> { using ZPZ = aerobus::zpz<367>; using type =
05251
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<361»; }; // NOLINT
           template<> struct ConwayPolynomial<367, 6> { using ZPZ = aerobus::zpz<367>; using type =
05253
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<321>, ZPZV<324>, ZPZV<6»; }; // NOLINT
05254
           template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type
      POLYY<ZPZV<1>, ZPZV<0>, 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 =
05255
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<335>, ZPZV<282>, ZPZV<50>, ZPZV<6»; };
05256
           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
05257
           template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<371»; }; // NOLINT
           template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2»; }; // NOLINT
05259
          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»; };
            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
           template<> struct ConwayPolynomial<373, 6> { using ZPZ = aerobus::zpz<373>; using type =
05262
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<83>, ZPZV<108>, ZPZV<20*; }; // NOLINT template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type
05263
      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
05264
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<203>, ZPZV<219>, ZPZV<66>, ZPZV<2»; };
05265
           template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<238>, ZPZV<370>, ZPZV<371»;
      }; // NOLINT
   template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<377»; }; // NOLINT
05267
           template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<379, 3> { using ZPZ = aerobus::zpz<379>; using type =
05268
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377»; }; // NOLINT
           template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327>, ZPZV<2»; }; // NOLINT
05270
           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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<374>, ZPZV<246>, ZPZV<246>, ZPZV<2»; }; // NOLINT
05271
           template<> struct ConwayPolynomial<379, 7> { using ZPZ = aerobus::zpz<379>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<377»; };
           template<> struct ConwayPolynomial<379, 8> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<210>, ZPZV<194>, ZPZV<173>, ZPZV<2»; }; //
      NOLINT
05274
           template<> struct ConwayPolynomial<379, 9> { using ZPZ = aerobus::zpz<379>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<36>>, ZPZV<365>, ZPZV<377»;
       }; // NOLINT
           template<> struct ConwayPolynomial<383, 1> { using ZPZ = aerobus::zpz<383>; using type =
05275
      POLYV<ZPZV<1>, ZPZV<378»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 2> { using ZPZ = aerobus::zpz<383>; using type =
05276
      POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
05278
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 5> { using ZPZ = aerobus::zpz<383>; using type =
05279
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
05280
           template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<8>, ZPZV<158>, ZPZV<5»; }; // NOLINT
05281
           template<> struct ConwayPolynomial<383, 7> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<378»; }; // NOLINT template<> struct ConwayPolynomial<383, 8> { using ZPZ = aerobus::zpz<383>; using type =
05282
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<281>, ZPZV<332>, ZPZV<296>, ZPZV<5»; }; //
       NOLINT
05283
           template<> struct ConwayPolynomial<383, 9> { using ZPZ = aerobus::zpz<383>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137>, ZPZV<16>, ZPZV<378»;
       }; // NOLINT
05284
           template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<387»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 2> { using ZPZ = aerobus::zpz<389>; using type =
05285
       POLYV<ZPZV<1>, ZPZV<379>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
05286
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<387»; }; // NOLINT template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
05287
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<266>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
05288
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<387»; }; // NOLINT
            template<> struct ConwayPolynomial<389, 6> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<339>, ZPZV<255>, ZPZV<2»; }; // NOLINT
05290
           template<> struct ConwayPolynomial<389, 7> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<387»; }; // NOLINT template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::zpz<389>; using type =
05291
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<351>, ZPZV<19>, ZPZV<290>, ZPZV<2»; };
       NOLINT
           template<> struct ConwayPolynomial<389, 9> { using ZPZ = aerobus::zpz<389>; using type =
05292
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<308>, ZPZV<387»;
       }; // NOLINT
05293
           template<> struct ConwayPolynomial<397, 1> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<392»; }; // NOLINT
           template<> struct ConwayPolynomial<397, 2> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5»; }; // NOLINT
05295
           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 =
05296
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392»; }; // NOLINT
05298
           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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<392»; }; // NOLINT
05299
```

```
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
        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<166>, ZPZV<166>, ZPZV<252>, ZPZV<392»;</pre>
05301
         }; // NOLINT
05302
               template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
         POLYV<ZPZV<1>, ZPZV<398»; }; // NOLINT
              template<> struct ConwayPolynomial<401, 2> { using ZPZ = aerobus::zpz<401>; using type =
05303
        POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3%; }; // NOLINT template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
05304
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<398»; }; // NOLINT template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type =
05305
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<3»; }; // NOLINT
05306
              template<> struct ConwayPolynomial<401, 5> { using ZPZ = aerobus::zpz<401>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<398»; }; // NOLINT template<> struct ConwayPolynomial<401, 6> { using ZPZ = aerobus::zpz<401>; using type =
05307
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<115>, ZPZV<51>, ZPZV<51>, ZPZV<30>; i, / NOLINT template<> struct ConwayPolynomial<401, 7> { using ZPZ = aerobus::zpz<401>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<398»; };
               template<> struct ConwayPolynomial<401, 8> { using ZPZ = aerobus::zpz<401>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<380>, ZPZV<113>, ZPZV<164>, ZPZV<3»; }; //
        template<> struct ConwayPolynomial<401, 9> { using ZPZ = aerobus::zpz<401>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<198>, ZPZV<398»;</pre>
05310
         }; // NOLINT
              template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
05311
         POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
               template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
05312
        POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
05313
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388»; }; // NOLINT
               template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
05314
        05315
               template<> struct ConwayPolynomial<409, 5> { using ZPZ = aerobus::zpz<409>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<388»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 6> { using ZPZ = aerobus::zpz<409>; using type =
05316
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<372>, ZPZV<53>, ZPZV<364>, ZPZV<21»; }; // NOLINT
05317
               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
05318
              template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<256>, ZPZV<69>, ZPZV<396>, ZPZV<396>, ZPZV<21»; }; //
         NOLINT
05319
              template<> struct ConwayPolynomial<409, 9> { using ZPZ = aerobus::zpz<409>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<81>, ZPZV<318>, ZPZV<211>, ZPZV<388»;
         }; // NOLINT
05320
              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 =
05321
         POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
05322
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<417»; }; // NOLINT
05323
               template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<373>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<419, 5> { using ZPZ = aerobus::zpz<419>; using type =
05324
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 6> { using ZPZ = aerobus::zpz<419>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<411>, ZPZV<33>, ZPZV<257>, ZPZV<2»; }; // NOLINT
               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<417»; }; // NOLIN template<> struct ConwayPolynomial<419, 8> { using ZPZ = aerobus::zpz<419>; using type =
05327
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<234>, ZPZV<388>, ZPZV<151>, ZPZV<2»; }; //
         NOLINT
               template<> struct ConwayPolynomial<419, 9> { using ZPZ = aerobus::zpz<419>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
         }; // NOLINT
05329
               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 =
05330
        POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
05331
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419»; }; // NOLINT template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
05332
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<421, 5> { using ZPZ = aerobus::zpz<421>; using type =
05333
        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 =
05334
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<111>, ZPZV<342>, ZPZV<41>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type =
05335
        POLYV-ZPZV-1>, ZPZV-(>, ZPZV-(>, ZPZV-(>), ZPZV-(>), ZPZV-(>), ZPZV-(>), ZPZV-(>), ZPZV-(21), ZPZV-(19); }; // NOLINT template<> struct ConwayPolynomial<421, 8> { using ZPZ = aerobus::zpz-421>; using type =
05336
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<389>, ZPZV<32>, ZPZV<77>, ZPZV<2»; };
05337
              template<> struct ConwayPolynomial<421, 9> { using ZPZ = aerobus::zpz<421>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<394>, ZPZV<145>, ZPZV<419»;
         }; // NOLINT
05338
              template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<424»; };
              template<> struct ConwayPolynomial<431, 2> { using ZPZ = aerobus::zpz<431>; using type =
        POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7»; }; // NOLINT
              template<> struct ConwayPolynomial<431, 3> { using ZPZ = aerobus::zpz<431>; using type =
05340
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424»; }; // NOLINT template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
05341
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7»; }; // NOLINT
05342
              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
05343
              template<> struct ConwayPolynomial<431, 6> { using ZPZ = aerobus::zpz<431>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<161>, ZPZV<202>, ZPZV<182>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial</a>431, 7> { using ZPZ = aerobus::zpz<431>; using type = DOLYMARPROADS
05344
        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
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<243>, ZPZV<286>, ZPZV<115>, ZPZV<7»; }; //
              template<> struct ConwayPolynomial<431, 9> { using ZPZ = aerobus::zpz<431>; using type =
05346
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<71>, ZPZV<329>, ZPZV<424»;
        }; // NOLINT
              template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
        POLYV<ZPZV<1>, ZPZV<428»; }; // NOLINT
              template<> struct ConwayPolynomial<433, 2> { using ZPZ = aerobus::zpz<433>; using type =
05348
        POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<433, 3> { using ZPZ = aerobus::zpz<433>; using type =
05349
        POLYV<ZPZV<1>, ZPZV<1>, 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
              template<> struct ConwayPolynomial<433, 5> { using ZPZ = aerobus::zpz<433>; using type =
05351
        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 =
05352
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<244>, ZPZV<353>, ZPZV<360>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<433, 7> { using ZPZ = aerobus::zpz<433>; using type
05353
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
05354
             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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<45>, ZPZV<45>, ZPZV<428»;
05355
        }; // NOLINT
05356
              template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
              template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
05357
        POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»: }: // NOLINT
05358
              template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424»; }; // NOLINT
05359
              template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; }; // NOLINT
              template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
05360
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT
              template<> struct ConwayPolynomial<439, 6> { using ZPZ = aerobus::zpz<439>; using type =
05361
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<324>, ZPZV<190>, ZPZV<15»; };
              template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT
05363
              template<> struct ConwayPolynomial<439, 8> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<359>, ZPZV<296>, ZPZV<266>, ZPZV<15»; }; //
        NOLINT
              template<> struct ConwayPolynomial<439, 9> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<342>, ZPZV<254>, ZPZV<424»;
        }; // NOLINT
05365
              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 =
05366
        POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41»; }; // NOLINT template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
05368
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<443>; using type =
05369
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT
05370
              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
05371
             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<6>, ZPZV<441»; }; // NOLINT template<> struct ConwayPolynomial<443, 8> { using ZPZ = aerobus::zpz<443>; using type =
05372
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<217>, ZPZV<290>, ZPZV<29; }; //
05373
              template<> struct ConwayPolynomial<4443, 9> { using ZPZ = aerobus::zpz<443>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<125>, ZPZV<109>, ZPZV<441»;
        }; // NOLINT
05374
              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
05376
             template<> struct ConwayPolynomial<449, 3> { using ZPZ = aerobus::zpz<449>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446»; }; // NOLINT template<> struct ConwayPolynomial<449, 4> { using ZPZ = aerobus::zpz<449>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3»; }; // NOLINT
05377
```

```
template<> struct ConwayPolynomial<449, 5> { using ZPZ = aerobus::zpz<449>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<446»; }; // NOLINT
05379
              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 =
05380
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<28>, ZPZV<446»; };
                                                                                                                                          // NOLINT
               template<> struct ConwayPolynomial<449, 8> { using ZPZ = aerobus::zpz<449>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361>, ZPZV<348>, ZPZV<124>, ZPZV<33»; }; //
         template<> struct ConwayPolynomial<449, 9> { using ZPZ = aerobus::zpz<449>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<446»; };
05382
         // NOLINT
05383
               template<> struct ConwayPolynomial<457, 1> { using ZPZ = aerobus::zpz<457>; using type =
         POLYV<ZPZV<1>, ZPZV<444»; }; // NOLINT
05384
              template<> struct ConwayPolynomial<457, 2> { using ZPZ = aerobus::zpz<457>; using type =
         POLYV<ZPZV<1>, ZPZV<454>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
05385
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444»; };
                                                                                 // NOLINT
               template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
05386
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13»; }; // NOLINT
               template<> struct ConwayPolynomial<457, 5> { using ZPZ = aerobus::zpz<457>; using type =
05387
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<444»; }; // NOLINT
               template<> struct ConwayPolynomial<457, 6> { using ZPZ = aerobus::zpz<457>; using type =
05388
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<205, ZPZV<266>, ZPZV<266>, ZPZV<257; dsing ZPZ template<> struct ConwayPolynomial<457, 7> { using ZPZ = aerobus::zpz<457>; using type
05389
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<444»; }; // NOLINT
05390
               template<> struct ConwayPolynomial<457, 8> { using ZPZ = aerobus::zpz<457>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<365>, ZPZV<296>, ZPZV<412>, ZPZV<13»; }; //
         NOLINT
05391
              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<9>, ZPZV<354>, ZPZV<844*;
         }; // NOLINT
               template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
05392
         POLYV<ZPZV<1>, ZPZV<459»; }; // NOLINT
               template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
05393
         POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<461, 3> { using ZPZ = aerobus::zpz<461>; using type =
05394
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459»; }; // NOLINT
05395
               template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
05396
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<459»; }; // NOLINT template<> struct ConwayPolynomial<461, 6> { using ZPZ = aerobus::zpz<461>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<439>, ZPZV<432>, ZPZV<329>, ZPZV<2»; }; // NOLINT
05397
               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<5 , ZPZV<5
05399
              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<32»; }; //
         NOLINT
               template<> struct ConwayPolynomial<461, 9> { using ZPZ = aerobus::zpz<461>; using type =
05400
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<210>, ZPZV<276>, ZPZV<459»;
         }; // NOLINT
05401
               template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
         POLYV<ZPZV<1>, ZPZV<460»; }; // NOLINT
               template<> struct ConwayPolynomial<463, 2> { using ZPZ = aerobus::zpz<463>; using type =
05402
         POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<10>, ZPZV<460»; }; // NOLINT template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
05404
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
05405
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<460»; }; // NOLINT
05406
               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
05407
              template<> struct ConwayPolynomial<463, 7> { using ZPZ = aerobus::zpz<463>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<460»; }; // NOLINT template<> struct ConwayPolynomial<463, 8> { using ZPZ = aerobus::zpz<463>; using type =
05408
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3414>, ZPZV<396>, ZPZV<39; //
         NOLINT
05409
               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
05410
               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 =
05411
         POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
05412
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
05413
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<467, 5> { using ZPZ = aerobus::zpz<467>; using type =
05414
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
               template<> struct ConwayPolynomial<467, 6> { using ZPZ = aerobus::zpz<467>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<123>, ZPZV<62>, ZPZV<237>, ZPZV<2»; }; // NOLINT
        template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<467, 8> { using ZPZ = aerobus::zpz<467>; using type =
05416
05417
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<318>, ZPZV<413>, ZPZV<289>, ZPZV<2*; }; //
05418
          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<397>, ZPZV<3447>, ZPZV<465»;
       }; // NOLINT
05419
           template<> struct ConwavPolynomial<479, 1> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<466»; }; // NOLINT
           template<> struct ConwayPolynomial<479, 2> { using ZPZ = aerobus::zpz<479>; using type =
05420
      POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; }; // NOLINT
05421
           template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466»; }; // NOLINT
           template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
05422
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13»; };
                                                                        // NOLINT
           template<> struct ConwayPolynomial<479, 5> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<466»; }; // NOLINT
05424
           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
05425
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<466»; }; // NOLINT
           template<> struct ConwayPolynomial<479, 8> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<247>, ZPZV<440>, ZPZV<17>, ZPZV<13»; }; //
      NOLINT
05427
           template<> struct ConwayPolynomial<479, 9> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<165>, ZPZV<466»; };
       // NOLINT
05428
           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 =
05429
      POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
05430
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; }; // NOLINT template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type =
05431
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3»; }; // NOLINT
          template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
05432
      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 =
05433
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<450>, ZPZV<185>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<487, 7> { using ZPZ = aerobus::zpz<487>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<484»; };
           template<> struct ConwayPolynomial<487, 8> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<249>, ZPZV<137>, ZPZV<3»; }; //
      NOLINT
           template<> struct ConwayPolynomial<487, 9> { using ZPZ = aerobus::zpz<487>; using type =
05436
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<271>, ZPZV<4447>, ZPZV<484»;
05437
           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 =
05438
      POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
05439
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type =
05440
      05441
           template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT
           template<> struct ConwayPolynomial<491, 6> { using ZPZ = aerobus::zpz<491>; using type =
05442
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<369>, ZPZV<402>, ZPZV<125>, ZPZV<2»; }; // NOLINT
                                                        7> { using ZPZ = aerobus::zpz<491>; using type
05443
           template<> struct ConwayPolynomial<491,
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 8> { using ZPZ = aerobus::zpz<491>; using type =
05444
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378>, ZPZV<372>, ZPZV<216>, ZPZV<2»; }; //
      NOLINT
05445
           template<> struct ConwayPolynomial<491, 9> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<453>, ZPZV<489»;
      }; // NOLINT
05446
           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 =
05447
      POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial4499, 3> { using ZPZ = aerobus::zpz<499>; using type =
05448
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
05449
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
05450
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 6> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<407>, ZPZV<191>, ZPZV<78>, ZPZV<7»; }; // NOLINT
           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<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 8> { using ZPZ = aerobus::zpz<499>; using type =
05453
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<288>, ZPZV<309>, ZPZV<200>, ZPZV<7»; }; //
      template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<491>, ZPZV<222>, ZPZV<492»;
      }; // NOLINT
           template<> struct ConwayPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
05455
      POLYV<ZPZV<1>, ZPZV<498»; }; // NOLINT
```

```
05456
           template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; }; // NOLINT
05457
           template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<498»; }; // NOLINT
template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<325>, ZPZV<5»; }; // NOLINT
05458
           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
           template<> struct ConwayPolynomial<503, 6> { using ZPZ = aerobus::zpz<503>; using type =
05460
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<380>, ZPZV<292>, ZPZV<255>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<503, 7> { using ZPZ = aerobus::zpz<503>; using type =
05461
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<498»; }; // NOLINT
           template<> struct ConwayPolynomial<503, 8> { using ZPZ = aerobus::zpz<503>; using type =
05462
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<441>, ZPZV<203>, ZPZV<316>, ZPZV<5»; };
       NOLINT
05463
           template<> struct ConwayPolynomial<503, 9> { using ZPZ = aerobus::zpz<503>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<158>, ZPZV<337>, ZPZV<498»;
       }; // NOLINT
05464
           template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<507»; }; // NOLINT
           template<> struct ConwayPolynomial<509, 2> { using ZPZ = aerobus::zpz<509>; using type =
       POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2»; }; // NOLINT
           05466
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
05467
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
05468
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
05469
           template<> struct ConwayPolynomial<509, 6> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<350>, ZPZV<232>, ZPZV<41>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<509, 7> { using ZPZ = aerobus::zpz<509>; using type struct convayPolynomial
05470
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<507»; };
           template<> struct ConwayPolynomial<509, 8> { using ZPZ = aerobus::zpz<509>; using type =
05471
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<420>, ZPZV<473>, ZPZV<382>, ZPZV<2»; }; //
       NOLINT
05472
           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<314>, ZPZV<28>, ZPZV<507»;
05473
           template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<518»; }; // NOLINT
05474
           template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
05475
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518»; }; // NOLINT template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<509>, ZPZV<3»; }; // NOLINT
05477
           template<> struct ConwayPolynomial<521, 5> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<518»; }; // NOLINT
           template<> struct ConwayPolynomial<521, 6> { using ZPZ = aerobus::zpz<521>; using type =
05478
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<315>, ZPZV<153>, ZPZV<280>, ZPZV<3»; }; // NOLINT
05479
           template<> struct ConwayPolynomial<521,
                                                       7> { using ZPZ = aerobus::zpz<521>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<518»; }; //
           template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type =
05480
       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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<181>, ZPZV<483>, ZPZV<518»;</pre>
05481
       }; // NOLINT
           template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
05482
      POLYV<ZPZV<1>, ZPZV<521»; }; // NOLINT
           template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
05483
      POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; }; // NOLINT
05484
           template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
05485
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
05486
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<521»; }; // NOLINT
           template<> struct ConwayPolynomial<523, 6> { using ZPZ = aerobus::zpz<523>; using type =
05487
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<475>, ZPZV<475>, ZPZV<371>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<523, 7> { using ZPZ = aerobus::zpz<523>; using type
05488
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<13>, ZPZV<13>, ZPZV<521»; };
05489
           template<> struct ConwayPolynomial<523, 8> { using ZPZ = aerobus::zpz<523>; using type =
       POLYV<2PZV<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<0>, ZPZV<19>, ZPZV<342>, ZPZV<145>, ZPZV<521»;
       }; // NOLINT
05491
           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 =
05492
      POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539»; };
                                                            // NOLINT
          template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
05494
      POLYV<ZPZV<1>, ZPZV<6>, ZPZV<65, ZPZV<333>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
05495
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<539»; };
      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 =
05497
      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 =
05498
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<376>, ZPZV<108>, ZPZV<113>, ZPZV<2»; }; //
05499
           template<> struct ConwayPolynomial<541, 9> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<340>, ZPZV<318>, ZPZV<539»;
       }; // NOLINT
            template<> struct ConwayPolynomial<547, 1> { using ZPZ = aerobus::zpz<547>; using type =
05500
      POLYV<ZPZV<1>, ZPZV<545»; }; // NOLINT
            template<> struct ConwayPolynomial<547, 2> { using ZPZ = aerobus::zpz<547>; using type =
       POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; }; // NOLINT
05502
           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 =
05503
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, 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<545»; }; // NOLINT
05505
           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<2x; }; // NOLINT template<> struct ConwayPolynomial<547, 7> { using ZPZ = aerobus::zpz<547>; using type
05506
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<545»; };
           template<> struct ConwayPolynomial<547, 8> { using ZPZ = aerobus::zpz<547>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2»; }; //
       NOLINT
      template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<263>, ZPZV<545»;</pre>
05508
       }; // NOLINT
05509
            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 =
05510
      POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
05511
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<555»; }; // NOLINT template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<557, 5> { using ZPZ = aerobus::zpz<557>; using type =
05513
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<555»; }; // NOLINT
           template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type =
0.5.51.4
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<202>, ZPZV<192>, ZPZV<253>, ZPZV<2»; }; // NOLINT
05515
           template<> struct ConwayPolynomial<557,
                                                          7> { using ZPZ = aerobus::zpz<557>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<555»; }; // NOLINT
05516
           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
05517
           template<> struct ConwayPolynomial<557, 9> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<1>, ZPZV<456>, ZPZV<434>, ZPZV<555»;
      }; // NOLINT template<> struct ConwayPolynomial<563, 1> { using ZPZ = aerobus::zpz<563>; using type =
05518
      POLYY<ZPZY<1>, ZPZY<561»; }; // NOLINT template<> struct ConwayPolynomial<563, 2> { using ZPZ = aerobus::zpz<563>; using type =
05519
      POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
05520
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; }; // NOLINT
           template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
05521
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<20>, ZPZY<399>, ZPZY<2*; }; // NOLINT template<> struct ConwayPolynomial<563, 5> { using ZPZ = aerobus::zpz<563>; using type =
05522
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<6>, ZPZY<6>, ZPZY<6>, ZPZY<6>, ZPZY<6>, ZPZY<6>, ZPZY<6>, ZPZY<6>, ZPZY<6>, ZPZY<66 , ZPZY<661s; }; // NOLINT template<> struct ConwayPolynomial<563, 6> { using ZPZ = aerobus::zpz<563>; using type =
05523
      POLYV<2PZV<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
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<561»; };
05525
          template<> struct ConwayPolynomial<563, 8> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<176>, ZPZV<509>, ZPZV<2»; }; //
       NOLINT
           template<> struct ConwayPolynomial<563, 9> { using ZPZ = aerobus::zpz<563>; using type =
05526
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<15>, ZPZV<19>, ZPZV<561»; };
05527
           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 =
05528
      POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
05530
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 5> { using ZPZ = aerobus::zpz<569>; using type =
05531
      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»; }; // NOLINI
05533
           template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<566»; }; // NOLINT template<> struct ConwayPolynomial<569, 8> { using ZPZ = aerobus::zpz<569>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<173>, ZPZV<241>, ZPZV<241>, ZPZV<3»; }; //
05534
```

```
NOLINT
         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
05536
               template<> struct ConwayPolynomial<571, 1> { using ZPZ = aerobus::zpz<571>; using type =
         POLYV<ZPZV<1>, ZPZV<568»; }; // NOLINT
               template<> struct ConwayPolynomial<571, 2> { using ZPZ = aerobus::zpz<571>; using type =
         POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
05538
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type =
05539
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<571, 5> { using ZPZ = aerobus::zpz<571>; using type =
05540
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<568»; }; // NOLINT
05541
              template<> struct ConwayPolynomial<571, 6> { using ZPZ = aerobus::zpz<571>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<221>, ZPZV<295>, ZPZV<3>; }; // NOLINT template<> struct ConwayPolynomial<571, 7> { using ZPZ = aerobus::zpz<571>; using type =
05542
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5,
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<363>, ZPZV<119>, ZPZV<371>, ZPZV<3*; }; //
05544
               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
05545
               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 =
05547
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572»; }; // NOLINT template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
05548
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<49; ); // NOLINT template<> struct ConwayPolynomial<577, 5> { using ZPZ = aerobus::zpz<577>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<572»; }; // NOLINT
05550
               template<> struct ConwayPolynomial<577, 6> { using ZPZ = aerobus::zpz<577>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<450>, ZPZV<25>, ZPZV<283>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<577, 7> { using ZPZ = aerobus::zpz<577>; using type =
05551
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<572»; }; //
               template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<3»; }; //
         NOLINT
         template<> struct ConwayPolynomial<577, 9> { using ZPZ = aerobus::zpz<577>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<576>, ZPZV<4449>, ZPZV<576>;
05553
         }; // NOLINT
               template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
         POLYV<ZPZV<1>, ZPZV<585»; }; // NOLINT
05555
              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 =
05556
         POLYY<ZPZY<1>, ZPZY<3>, ZPZY<3>, ZPZY<585»; }; // NOLINT template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
05557
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<444>, ZPZV<2»; }; // NOLINT
05558
               template<> struct ConwayPolynomial<587, 5> { using ZPZ = aerobus::zpz<587>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<585»; }; // NOLINT template<> struct ConwayPolynomial<587, 6> { using ZPZ = aerobus::zpz<587>; using type =
05559
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<204, ZPZV<204, ZPZV<26>, ZPZV<226>, ZPZV<2»; ); // NOLINT template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<585»; };
              template<> struct ConwayPolynomial<587, 8> { using ZPZ = aerobus::zpz<587>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<492>, ZPZV<44>, ZPZV<91>, ZPZV<91>, ZPZV<2»; };
         NOLINT
         template<> struct ConwayPolynomial<587, 9> { using ZPZ = aerobus::zpz<587>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<333>, ZPZV<55>, ZPZV<585»;
05562
              template<> struct ConwayPolynomial<593, 1> { using ZPZ = aerobus::zpz<593>; using type =
05563
         POLYV<ZPZV<1>, ZPZV<590»; }; // NOLINT
05564
               template<> struct ConwayPolynomial<593, 2> { using ZPZ = aerobus::zpz<593>; using type =
         POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<593, 3> { using ZPZ = aerobus::zpz<593>; using type =
05565
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<65, ZPZV<590»; }; // NOLINT template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
05566
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<593, 5> { using ZPZ = aerobus::zpz<593>; using type =
05567
         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 =
05568
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<345>, ZPZV<65>, ZPZV<478>, ZPZV<3»; }; // NOLINI
               template<> struct ConwayPolynomial<593, 7> { using ZPZ = aerobus::zpz<593>; using type
05569
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<590»; }; // NOLINT template<> struct ConwayPolynomial<593, 8> { using ZPZ = aerobus::zpz<593>; using type =
05570
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<350>, ZPZV<291>, ZPZV<495>, ZPZV<3»; }; //
         NOLINT
05571
               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<0>, ZPZV<2>3>, ZPZV<223>, ZPZV<523>, ZPZV<590»;
         }; // NOLINT
05572
               \texttt{template<> struct ConwayPolynomial<599, 1> \{ using ZPZ = aerobus::zpz<599>; using type = 1 \} 
        POLYV<ZPZV<1>, ZPZV<592»; }; // NOLINT
              template<> struct ConwayPolynomial<599, 2> { using ZPZ = aerobus::zpz<599>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; };
               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 =
05575
                                                                                              // NOLINT
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<599, 5> { using ZPZ = aerobus::zpz<599>; using type =
05576
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<592»; }; // NOLINT
05577
               template<> struct ConwayPolynomial<599, 6> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type =
05578
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZV<0
05579
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<440>, ZPZV<37>, ZPZV<124>, ZPZV<124>, ZPZV<7»; }; //
05580
              template<> struct ConwayPolynomial<599, 9> { using ZPZ = aerobus::zpz<599>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<9>, ZPZV<592»;
         }; // NOLINT
05581
               template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type =
         POLYV<ZPZV<1>, ZPZV<594»; }; // NOLINT
               template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
         POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
05583
               template<> struct ConwayPolynomial<601, 3> { using ZPZ = aerobus::zpz<601>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<594»; }; // NOLINT
template<> struct ConwayPolynomial<601, 4> { using ZPZ = aerobus::zpz<601>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; }; // NOLINT
05584
               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
               template<> struct ConwayPolynomial<601, 6> { using ZPZ = aerobus::zpz<601>; using type =
05586
         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 =
05587
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<5), ZPZV<5), ZPZV<5); // NOLINT template<> struct ConwayPolynomial<601, 8> { using ZPZ = aerobus::zpz<601>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>0>, ZPZV<550>, ZPZV<241>, ZPZV<490>, ZPZV<490>, //
        template<> struct ConwayPolynomial<601, 9> { using ZPZ = aerobus::zpz<601>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<487>, ZPZV<487>, ZPZV<594»;</pre>
05589
         }; // NOLINT
               template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<604»; }; // NOLINT
              template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
05591
        POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
05592
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<604»; }; // NOLINT template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
05593
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3»; }; // NOLINT
05594
               template<> struct ConwayPolynomial<607, 5> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<604»; }; // NOLINT
05595
              template<> struct ConwayPolynomial<607, 6> { using ZPZ = aerobus::zpz<607>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<45>, ZPZV<478>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type =
05596
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<604»; };
               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*; };
              template<> struct ConwayPolynomial<607, 9> { using ZPZ = aerobus::zpz<607>; using type =
05598
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<444>, ZPZV<429>, ZPZV<604»;
         }; // NOLINT
   template<> struct ConwayPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
05599
         POLYV<ZPZV<1>, ZPZV<611»; }; // NOLINT
05600
               template<> struct ConwayPolynomial<613, 2> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
05601
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<611»; }; // NOLINT
               template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2»; }; // NOLINT
05603
              template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
         template<> struct ConwayPolynomial<613, 6> { using ZPZ = aerobus::zpz<613>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<609>, ZPZV<595>, ZPZV<601>, ZPZV<2»; }; // NOLINT
05604
               template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<611»; }; // NOLIN template<> struct ConwayPolynomial<613, 8> { using ZPZ = aerobus::zpz<613>; using type =
05606
         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 =
05607
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<536>, ZPZV<611»;
         }; // NOLINT
05608
               template<> struct ConwayPolynomial<617, 1> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<614»; }; // NOLINT
05609
               template<> struct ConwayPolynomial<617, 2> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
05610
        POLYV<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<614»; }; // NOLINT template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type =
05611
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<617, 5> { using ZPZ = aerobus::zpz<617>; using type =
05612
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<614»; }; // NOLINT
```

```
05613
                        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 =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<614»; }; // NOLINT template<> struct ConwayPolynomial<617, 8> { using ZPZ = aerobus::zpz<617>; using type =
05615
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<501>, ZPZV<501>, ZPZV<155>, ZPZV<3»; }; //
              NOLINT
                        template<> struct ConwayPolynomial<617, 9> { using ZPZ = aerobus::zpz<617>; using type
05616
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<543>, ZPZV<614»;
              }; // NOLINT
05617
                       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 =
05618
              POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; }; // NOLINT
05619
                      template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617»; }; // NOLINT
template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<619, 5> { using ZPZ = aerobus::zpz<619>; using type =
05620
05621
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<617»; }; // NOLINT
                        template<> struct ConwayPolynomial<619, 6> { using ZPZ = aerobus::zpz<619>; using type =
              POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<468>, ZPZV<347>, ZPZV<2»; }; // NOLINT
05623
                      template<> struct ConwayPolynomial<619, 7> { using ZPZ = aerobus::zpz<619>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZV<0>; ZPZV<0
05624
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0), ZPZV<10>, ZPZV<416>, ZPZV<383>, ZPZV<225>, ZPZV<2»; }; //
              template<> struct ConwayPolynomial<619, 9> { using ZPZ = aerobus::zpz<619>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<579>, ZPZV<510>, ZPZV<617»;
05625
              }; // NOLINT
                       template<> struct ConwayPolynomial<631, 1> { using ZPZ = aerobus::zpz<631>; using type =
05626
              POLYV<ZPZV<1>, ZPZV<628»; }; // NOLINT
                       template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
05627
              POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3»; }; // NOLINT
05628
                        template<> struct ConwayPolynomial<631, 3> { using ZPZ = aerobus::zpz<631>; using type =
              POLYY<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
05629
05630
                        template<> struct ConwayPolynomial<631, 5> { using ZPZ = aerobus::zpz<631>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
              template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
POLYV<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 =
05631
05632
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       template<> struct ConwayPolynomial<631, 8> { using ZPZ = aerobus::zpz<631>; using type
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<3»; }; //
              NOLINT
              template<> struct ConwayPolynomial<631, 9> { using ZPZ = aerobus::zpz<631>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<296>, ZPZV<413>, ZPZV<413>, ZPZV<413>, ZPZV<413>, ZPZV<413</pre>
05634
              }; // NOLINT
05635
                        template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
              POLYV<ZPZV<1>, ZPZV<638»; }; // NOLINT
05636
                       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 =
05637
              POLYY<ZPZY<1>, ZPZV<0>, ZPZV<4>, ZPZV<638»; }; // NOLINT template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<3»; }; // NOLINT
                        template<> struct ConwayPolynomial<641, 5> { using ZPZ = aerobus::zpz<641>; using type =
05639
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<638»; }; // NOLINT
                       template<> struct ConwayPolynomial<641, 6> { using ZPZ = aerobus::zpz<641>; using type =
05640
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<294>, ZPZV<294>, ZPZV<294>; ZPZV<294>; ZPZV<3*; }; // NOLINT template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type
05641
              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<638»; }; //
05642
                       template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<3»; }; //
              NOLINT
05643
                      template<> struct ConwayPolynomial<641, 9> { using ZPZ = aerobus::zpz<641>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
              }; // NOLINT
                       template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
05644
              POLYV<ZPZV<1>, ZPZV<632»; }; // NOLINT
                       template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
05645
              POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11»; }; // NOLINT
              template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<632»; }; // NOLINT
05646
                       template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
05647
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<600>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<643, 5> { using ZPZ = aerobus::zpz<643>; using type =
05648
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<632»; }; // NOLINT
                       template<> struct ConwayPolynomial<643, 6> { using ZPZ = aerobus::zpz<643>; using type =
05649
              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
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<632»; };
05651
                      template<> struct ConwayPolynomial<643, 8> { using ZPZ = aerobus::zpz<643>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<631>, ZPZV<573>, ZPZV<569>, ZPZV<11»; }; //
              NOLTNT
```

```
template<> struct ConwayPolynomial<643, 9> { using ZPZ = aerobus::zpz<643>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<591>, ZPZV<475>, ZPZV<632»;
        }; // NOLINT
05653
              template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<642»; }; // NOLINT
              template<> struct ConwayPolynomial<647, 2> { using ZPZ = aerobus::zpz<647>; using type =
05654
        POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<647, 3> { using ZPZ = aerobus::zpz<647>; using type =
05655
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<642»; }; // NOLINT template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
05656
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<647, 5> { using ZPZ = aerobus::zpz<647>; using type =
05657
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<642»; }; // NOLINT
               template<> struct ConwayPolynomial<647, 6> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<308>, ZPZV<385>, ZPZV<642>, ZPZV<5»; };
05659
              template<> struct ConwayPolynomial<647, 7> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<642»; }; // NOLINT template<> struct ConwayPolynomial<647, 8> { using ZPZ = aerobus::zpz<647>; using type =
05660
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<603>, ZPZV<259>, ZPZV<271>, ZPZV<5»; }; //
        template<> struct ConwayPolynomial<647, 9> { using ZPZ = aerobus::zpz<647>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<561>, ZPZV<561>, ZPZV<123>, ZPZV<642»;
        }; // NOLINT
              template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
05662
        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
05664
              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 =
05665
        POLYVCZPZV<1>, ZPZV<6>, ZPZV<65, ZPZV<596>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
05666
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651»; }; // NOLINT
05667
             template<> struct ConwayPolynomial<653, 6> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<45>, ZPZV<220>, ZPZV<242>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type =
05668
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<651»; }; // NOLINT
              template<> struct ConwayPolynomial<653, 8> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<385>, ZPZV<18>, ZPZV<296>, ZPZV<2»; };
05670
             template<> struct ConwayPolynomial<653, 9> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<60>, ZPZV<0>, ZPZ
        }; // NOLINT
05671
              template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<657»; }; // NOLINT
05672
              template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2»; }; // NOLINT
05673
              template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<657»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
05674
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
05675
        05676
              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<22»; }; // NOLINT template<> struct ConwayPolynomial<659, 7> { using ZPZ = aerobus::zpz<659>; using type
05677
        POLYV<2PZV<1>, 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 =
05678
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<358>, ZPZV<246>, ZPZV<90>, ZPZV<20; };
        NOLINT
05679
              template<> struct ConwayPolynomial<659, 9> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592>, ZPZV<5657»;
        }; // NOLINT
               template<> struct ConwayPolynomial<661, 1> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<659»; }; // NOLINT
05681
              template<> struct ConwayPolynomial<661, 2> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
05682
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<659»; }; // NOLINT
              template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type =
05683
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2»; }; // NOLINT
             template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
05684
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<659»; }; // NOLINT template<> struct ConwayPolynomial<661, 6> { using ZPZ = aerobus::zpz<661>; using type =
05685
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<382>, ZPZV<2»; }; // NOLINT 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<0>, ZPZV<2>, ZPZV<659»; };
05687
              template<> struct ConwayPolynomial<661, 8> { using ZPZ = aerobus::zpz<661>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<612>, ZPZV<285>, ZPZV<72>, ZPZV<2»; }; //
        NOLINT
              template<> struct ConwayPolynomial<661, 9> { using ZPZ = aerobus::zpz<661>; using type =
05688
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<389>, ZPZV<220>, ZPZV<659»;
        }; // NOLINT
05689
              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 =
05690
        POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5»; }; // NOLINT
```

```
05691
           template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
05692
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<673, 5> { using ZPZ = aerobus::zpz<673>; using type =
05693
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<668»; }; // NOLINT
           template<> struct ConwayPolynomial<673, 6> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<524>, ZPZV<248>, ZPZV<35>, ZPZV<5»; }; // NOLINI
          template<> struct ConwayPolynomial<673, 7> { using ZPZ = aerobus::zpz<673>; using type =
05695
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<668»; }; // NOLINT
05696
          template<> struct ConwayPolynomial<673, 8> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<587>, ZPZV<302>, ZPZV<5»; }; //
           template<> struct ConwayPolynomial<673, 9> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<347>, ZPZV<553>, ZPZV<668»;
      }; // NOLINT
           template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
05698
      POLYV<ZPZV<1>, ZPZV<675»; }; // NOLINT
           template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<672>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<677, 3> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<675»; }; // NOLINT template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
05701
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<677, 5> { using ZPZ = aerobus::zpz<677>; using type =
05702
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<675»; }; // NOLINT
05703
           template<> struct ConwayPolynomial<677, 6> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446>, ZPZV<632>, ZPZV<50>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type :
05704
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<675»; }; // NOLINT
          template<> struct ConwayPolynomial<677, 8> { using ZPZ = aerobus::zpz<677>; using type =
05705
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3619>, ZPZV<152>, ZPZV<152>, ZPZV<2»; }; //
05706
          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<404>, ZPZV<675»;
      }; // NOLINT
      template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type = POLYV<ZPZV<1>, ZPZV<678»; }; // NOLINT
05707
           template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; }; // NOLINT
05709
          template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<678»; }; // NOLINT
template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<45>, ZPZV<5»; }; // NOLINT
0.5710
           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
05712
          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 =
05713
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<678»; }; // NOLINT
           template<> struct ConwayPolynomial<683, 8> { using ZPZ = aerobus::zpz<683>; using type
05714
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<383>, ZPZV<184>, ZPZV<65>, ZPZV<5»; };
      NOLINT
      template<> struct ConwayPolynomial<683, 9> { using ZPZ = aerobus::zpz<683>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<678>;
05715
      }; // NOLINT
05716
           template<> struct ConwayPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<688»; }; // NOLINT
           template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
05717
      POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
05718
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<688»; }; // NOLINT template<> struct ConwayPolynomial<691, 4> { using ZPZ = aerobus::zpz<691>; using type =
05719
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; }; // NOLINT
05720
          template<> struct ConwayPolynomial<691, 5> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; }; // NOLINT
05721
           template<> struct ConwayPolynomial<691, 6> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<262>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<688»; }; //
           template<> struct ConwayPolynomial<691, 8> { using ZPZ = aerobus::zpz<691>; using type
05723
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<356>, ZPZV<425>, ZPZV<321>, ZPZV<3»; }; //
      NOLINT
05724
           template<> struct ConwayPolynomial<691, 9> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<556>, ZPZV<443>, ZPZV<688»;
      }; // NOLINT
05725
           template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<699»; }; // NOLINT
           template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
05726
      POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
05727
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; }; // NOLINT
           template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2»; }; // NOLINT
05729
          template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699»; }; // NOLINT
05730
          template<> struct ConwayPolynomial<701, 6> { using ZPZ = aerobus::zpz<701>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<571>, ZPZV<327>, ZPZV<285>, ZPZV<2»; };
                  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<0>, ZPZV<10>, ZPZV<699»; };
                                                                                                                                                                 // NOLINT
                template<> struct ConwayPolynomial<701, 8> { using ZPZ = aerobus::zpz<701>; using type =
05732
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<619>, ZPZV<206>, ZPZV<593>, ZPZV<2w; }; //
          NOLINT
                 template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<4>, ZPZV<45>, ZPZV<459>, ZPZV<373>, ZPZV<699»;
          }; // NOLINT
05734
                 template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
          POLYV<ZPZV<1>, ZPZV<707»; }; // NOLINT
                 05735
          POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707%; }; // NOLINT template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
05737
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<709, 5> { using ZPZ = aerobus::zpz<709>; using type =
05738
          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<6>, ZPZV<66>, ZPZV<514>, ZPZV<295>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<709, 7> { using ZPZ = aerobus::zpz<709>; using type =
05740
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<707%; }; // NOLINT template<> struct ConwayPolynomial<709, 8> { using ZPZ = aerobus::zpz<709>; using type =
0.5741
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<689>, ZPZV<233>, ZPZV<79>, ZPZV<2»; };
                 template<> struct ConwayPolynomial<709, 9> { using ZPZ = aerobus::zpz<709>; using type =
05742
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZP
          }; // NOLINT
05743
                 template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
          POLYV<ZPZV<1>, ZPZV<708»; }; // NOLINT
05744
                  template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
          POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; }; // NOLINT
05745
                template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
05746
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<5>, ZPZV<602>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<719, 5> { using ZPZ = aerobus::zpz<719>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
                 template<> struct ConwayPolynomial<719, 6> { using ZPZ = aerobus::zpz<719>; using type =
05748
          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
05749
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<708»; }; // NOLINT
                 template<> struct ConwayPolynomial<719, 8> { using ZPZ = aerobus::zpz<719>; using type =
05750
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<714>, ZPZV<362>, ZPZV<244>, ZPZV<11»; }; //
          NOLINT
05751
                 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
05752
                  template<> struct ConwavPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
          POLYV<ZPZV<1>, ZPZV<722»; }; // NOLINT
                  template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
          POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; }; // NOLINT
                 template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
05754
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<7>; }; // NOLINT template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
05755
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5»; };
                                                                                                              // NOLINT
                 template<> struct ConwayPolynomial<727, 5> { using ZPZ = aerobus::zpz<727>; using type =
05756
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<722»; }; // NOLINT
05757
                  template<> struct ConwayPolynomial<727, 6> { using ZPZ = aerobus::zpz<727>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<86>, ZPZV<397>, ZPZV<672>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<727, 7> { using ZPZ = aerobus::zpz<727>; using type = aerobus::zpz<7
05758
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<12»; }; // NOLINT
                 template<> struct ConwayPolynomial<727, 8> { using ZPZ = aerobus::zpz<727>;
                                                                                                                                                        using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<639>, ZPZV<671>, ZPZV<368>, ZPZV<5»; }; //
          NOLINT
05760
          template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<573>, ZPZV<502>, ZPZV<722»;</pre>
          }; // NOLINT
05761
                  template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
          POLYV<ZPZV<1>, ZPZV<727»; }; // NOLINT
05762
                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 =
05763
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<4>, ZPZV<727%; }; // NOLINT template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<6»; }; // NOLINT
05765
                 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
05766
                 template<> struct ConwayPolynomial<733, 6> { using ZPZ = aerobus::zpz<733>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<549>, ZPZV<151>, ZPZV<6»; }; // NOLINT
                 template<> struct ConwayPolynomial<733,
                                                                                       7> { using ZPZ = aerobus::zpz<733>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<727%; }; // NOLI template<> struct ConwayPolynomial<733, 8> { using ZPZ = aerobus::zpz<733>; using type
05768
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<532>, ZPZV<610>, ZPZV<142>, ZPZV<6%; }; //
05769
                 template<> struct ConwayPolynomial<733, 9> { using ZPZ = aerobus::zpz<733>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<337>, ZPZV<6>, ZPZV<727»; };
05770
          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 =
0.5771
      POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; }; // NOLINT
          template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
05773
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
05774
      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 =
05775
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<447>, ZPZV<625>, ZPZV<63»; }; // NOLIN
05776
          template<> struct ConwayPolynomial<739, 7> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<44>, ZPZV<736»; }; // NOLINT template<> struct ConwayPolynomial<739, 8> { using ZPZ = aerobus::zpz<739>; using type =
05777
      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<1>, ZPZV<616>, ZPZV<81>, ZPZV<736»;
      }; // NOLINT
05779
           template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<738»; }; // NOLINT
05780
           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 =
05781
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<738»; }; // NOLINT template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
05782
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<743, 5> { using ZPZ = aerobus::zpz<743>; using type =
05783
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT
           template<> struct ConwayPolynomial<743, 6> { using ZPZ = aerobus::zpz<743>; using type =
05784
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<236>, ZPZV<471>, ZPZV<88>, ZPZV<5»; }; // NOLINT
05785
           template<> struct ConwayPolynomial<743, 7> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT
           template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
05786
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<279>, ZPZV<588>, ZPZV<58»; }; //
      NOLINT
          template<> struct ConwayPolynomial<743, 9> { using ZPZ = aerobus::zpz<743>; using type =
05787
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<327>, ZPZV<6765, ZPZV<738»;
      }; // NOLINT
           template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
05788
      POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
           template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT
05790
          template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; ); // NOLINT
template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<525>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<751, 5> { using ZPZ = aerobus::zpz<751>; using type =
05791
05792
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT
05793
           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<539>, ZPZV<53; }; // NOLINT template<> struct ConwayPolynomial<751, 7> { using ZPZ = aerobus::zpz<751>; using type =
05794
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>; // NOLINT template<> struct ConwayPolynomial<751, 8> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<741>, ZPZV<243>, ZPZV<672>, ZPZV<3»; }; //
05796
           template<> struct ConwayPolynomial<751, 9> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<748»;
      }; // NOLINT
05797
           template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
05798
          template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2»; }; // NOLINT
05799
           template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
      POLYVCZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<65, ZPZV<755»; j; // NOLINT template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
05800
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; };
                                                                       // NOLINT
           template<> struct ConwayPolynomial<757, 5> { using ZPZ = aerobus::zpz<757>; using type =
05801
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<755»; }; // NOLINT
05802
           template<> struct ConwayPolynomial<757, 6> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<753>, ZPZV<739>, ZPZV<745>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<757,
                                                        7> { using ZPZ = aerobus::zpz<757>; using type
05803
      POLYV<2PZV<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 =
05804
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<110>, ZPZV<509>, ZPZV<2»; }; //
      NOLINT
05805
      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<688>, ZPZV<688>, ZPZV<608>, ZPZV<755»;</pre>
      }; // NOLINT
    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 =
05807
      POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; }; // NOLINT
          template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755»; }; // NOLINT
       template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<6»; }; // NOLINT
           template<> struct ConwayPolynomial<761, 5> { using ZPZ = aerobus::zpz<761>; using type =
0.5810
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT
       template<> struct ConwayPolynomial<761, 6> { using ZPZ = aerobus::zpz<761>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<634>, ZPZV<597>, ZPZV<155>, ZPZV<6»; }; // NOLINT
05811
            template<> struct ConwayPolynomial<761, 7> { using ZPZ = aerobus::zpz<761>; using type
05812
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<761, 8> { using ZPZ = aerobus::zpz<761>; using type =
05813
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<603>, ZPZV<144>, ZPZV<540>, ZPZV<5%; }; //
       NOLINT
           template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type =
05814
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<317>, ZPZV<571>, ZPZV<755»;
       }; // NOLINT
05815
            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 =
05816
       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 template<> struct ConwayPolynomial<769, 4> { using ZPZ = aerobus::zpz<769>; using type =
05818
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<769, 5> { using ZPZ = aerobus::zpz<769>; using type =
0.5819
       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<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<326>, ZPZV<650>, ZPZV<11»; }; // NOLINI
05821
            template<> struct ConwayPolynomial<769, 7> { using ZPZ = aerobus::zpz<769>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<758»; }; // NOLINT
05822
           template<> struct ConwayPolynomial<769, 8> { using ZPZ = aerobus::zpz<769>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<560>, ZPZV<574>, ZPZV<632>, ZPZV<11»; }; //
       NOLINT
            template<> struct ConwayPolynomial<769, 9> { using ZPZ = aerobus::zpz<769>; using type =
05823
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<623>, ZPZV<751>, ZPZV<758»;
       }; // NOLINT
05824
            template<> struct ConwayPolynomial<773, 1> { using ZPZ = aerobus::zpz<773>; using type =
       POLYV<ZPZV<1>, ZPZV<771»; }; // NOLINT
            template<> struct ConwayPolynomial<773, 2> { using ZPZ = aerobus::zpz<773>; using type =
       POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
05826
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<771»; }; // NOLINT template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
05827
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<773, 5> { using ZPZ = aerobus::zpz<773>; using type =
05828
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<771»; }; // NOLINT
05829
            template<> struct ConwayPolynomial<773, 6> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<91>, ZPZV<3>, ZPZV<581>, ZPZV<581>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<773, 7> { using ZPZ = aerobus::zpz<773>; using type =
05830
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<771»; }; // NOLINT
           template<> struct ConwayPolynomial<773, 8> { using ZPZ = aerobus::zpz<773>; using type =
05831
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<484>, ZPZV<94>, ZPZV<693>, ZPZV<693; }; //
05832
           template<> struct ConwayPolynomial<773, 9> { using ZPZ = aerobus::zpz<773>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<216>, ZPZV<574>, ZPZV<771»;
       }; // NOLINT
05833
            template<> struct ConwayPolynomial<787, 1> { using ZPZ = aerobus::zpz<787>; using type =
       POLYV<ZPZV<1>, ZPZV<785»; }; // NOLINT
           template<> struct ConwayPolynomial<787, 2> { using ZPZ = aerobus::zpz<787>; using type =
       POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
05835
            template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785»; ); // NOLINT
template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2»; }; // NOLINT
05836
            template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<785»; }; // NOLINT
05838
           template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<98>, ZPZV<512>, ZPZV<606>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<787, 7> { using ZPZ = aerobus::zpz<787>; using type =
05839
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<38>, ; // NOLINT
            template<> struct ConwayPolynomial<787, 8> { using ZPZ = aerobus::zpz<787>; using type
05840
       POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<0>, 2PZV<5>, 2PZV<612>, 2PZV<26>, 2PZV<715>, 2PZV<2»; };
       template<> struct ConwayPolynomial<787, 9> { using ZPZ = aerobus::zpz<787>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<573>, ZPZV<785»;</pre>
05841
       }; // NOLINT
05842
            template<> struct ConwayPolynomial<797, 1> { using ZPZ = aerobus::zpz<797>; using type =
       POLYV<ZPZV<1>, ZPZV<795»; }; // NOLINT
05843
            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 =
05844
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type =
05845
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<1717>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<797, 5> { using ZPZ = aerobus::zpz<797>; using type =
05846
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 6> { using ZPZ = aerobus::zpz<797>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<657>, ZPZV<396>, ZPZV<71>, ZPZV<2»; }; // NOLINT
05847
```

```
template<> struct ConwayPolynomial<797, 7> { using ZPZ = aerobus::zpz<797>; using type
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type =
05849
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2»; }; //
         NOLINT
              template<> struct ConwayPolynomial<797, 9> { using ZPZ = aerobus::zpz<797>; using type =
05850
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<240>, ZPZV<240>, ZPZV<599>, ZPZV<795»;
         }; // NOLINT
05851
              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 =
05852
         POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
05853
              template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<806»; }; // NOLINT
05854
              template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<809, 5> { using ZPZ = aerobus::zpz<809>; using type =
05855
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; }; // NOLINT
              template<> struct ConwayPolynomial<809, 6> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<562>, ZPZV<75>, ZPZV<43>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<809, 7> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<806»; }; // NOLINT
05858
              template<> struct ConwayPolynomial<809, 8> { using ZPZ = aerobus::zpz<809>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<593>, ZPZV<745>, ZPZV<673>, ZPZV<63»; }; //
         NOLINT
               template<> struct ConwayPolynomial<809, 9> { using ZPZ = aerobus::zpz<809>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<341>, ZPZV<727>, ZPZV<806»;
         }; // NOLINT
05860
               template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
        POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
              template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
05861
        POLYV<ZPZV<1>, ZPZV<806>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<811, 3> { using ZPZ = aerobus::zpz<811>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<808»; }; // NOLINT
              template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type =
05863
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<811, 5> { using ZPZ = aerobus::zpz<811>; using type =
05864
        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 =
        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 =
05866
        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 =
05867
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<5.
05868
              template<> struct ConwayPolynomial<811, 9> { using ZPZ = aerobus::zpz<811>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<382>, ZPZV<200>, ZPZV<808»;
         }; // NOLINT
05869
              template<> struct ConwavPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<819»; }; // NOLINT
              template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
05870
        POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT
05871
              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 =
05872
        POLYV<ZPZV<1>, ZPZV<6>, ZPZV<662>, ZPZV<60>, ZPZV<60>, ZPZV<2>; ); // NOLINT template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<819»; }; // NOLINT
              template<> struct ConwayPolynomial<821, 6> { using ZPZ = aerobus::zpz<821>; using type =
05874
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<160>, ZPZV<130>, ZPZV<803>, ZPZV<2»; }; // NOLINT
05875
              template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, 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»; }; //
        template<> struct ConwayPolynomial<821, 9> { using ZPZ = aerobus::zpz<821>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<650>, ZPZV<650>, ZPZV<557>, ZPZV<819»;</pre>
05877
         }; // NOLINT
               template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
05878
        POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
              template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
05880
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
  template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT</pre>
05881
              template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
05882
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
05883
              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 = DOLYVZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0 + Z
05884
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<820»; };
              template<> struct ConwayPolynomial<823, 8> { using ZPZ = aerobus::zpz<823>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<451>, ZPZV<437>, ZPZV<31>, ZPZV<3»; };
        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>
05886
```

```
}; // NOLINT
05887
           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 =
05888
      POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
05889
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825»; }; // NOLINT
           template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type =
05890
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<827, 5> { using ZPZ = aerobus::zpz<827>; using type =
05891
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<825»; }; // NOLINT template<> struct ConwayPolynomial<827, 6> { using ZPZ = aerobus::zpz<827>; using type =
05892
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<085>, ZPZV<685>, ZPZV<691>, ZPZV<691>, ZPZV<29; }; // NOLINT template<> struct ConwayPolynomial<827, 7> { using ZPZ = aerobus::zpz<827>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<825»; }; // NOLINT
05894
           template<> struct ConwayPolynomial<827, 8> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<812>, ZPZV<79>, ZPZV<32>, ZPZV<2»; };
      NOLINT
05895
           template<> struct ConwayPolynomial<827, 9> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<372>, ZPZV<825»;
      }; // NOLINT
05896
           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 =
05897
      POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<827»; }; // NOLINT
           template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
05899
      template<> struct ConwayPolynomial<829, 5> { using ZPZ = aerobus::zpz<829>; using type =
05900
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<827»; }; // NOLINT
05901
           template<> struct ConwayPolynomial<829, 6> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<476>, ZPZV<817>, ZPZV<2»; }; // NOLINT
05902
          template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<827»; }; // NOLINT template<> struct ConwayPolynomial<829, 8> { using ZPZ = aerobus::zpz<829>; using type =
05903
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<468>, ZPZV<241>, ZPZV<138>, ZPZV<2»; }; //
05904
           template<> struct ConwayPolynomial<829, 9> { using ZPZ = aerobus::zpz<829>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<621>, ZPZV<552>, ZPZV<827»;
      }; // NOLINT
05905
           template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<828»; }; // NOLINT
05906
           template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; }; // NOLINT
05907
           template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; }; // NOLINT template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11»; }; // NOLINT
05908
           template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
05909
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<28»; }; // NOLINT
           template<> struct ConwayPolynomial<839, 6> { using ZPZ = aerobus::zpz<839>; using type =
05910
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<370>, ZPZV<537>, ZPZV<23>, ZPZV<11»; }; // NOLINT
05911
           template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<828»; }; // NOLINT
           template<> struct ConwayPolynomial<839, 8> { using ZPZ = aerobus::zpz<839>; using type
05912
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<553>, ZPZV<779>, ZPZV<329>, ZPZV<11»; }; //
      template<> struct ConwayPolynomial<839, 9> { using ZPZ = aerobus::zpz<839>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<349>, ZPZV<206>, ZPZV<828*;
05913
      }; // NOLINT
05914
           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
05916
           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 =
05917
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
05918
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<851»; }; // NOLINT
05919
           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 =
05920
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<45); }; // NOLIN template<> struct ConwayPolynomial<853, 8> { using ZPZ = aerobus::zpz<853>; using type =
                                                                                                      // NOLINT
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<544>, ZPZV<846>, ZPZV<118>, ZPZV<2»; };
05922
           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<1>, ZPZV<677>, ZPZV<621>, ZPZV<821>,
       }; // NOLINT
           template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
      POLYV<ZPZV<1>, ZPZV<854»; }; // NOLINT
05924
           template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
      POLYV<ZPZV<1>, ZPZV<85>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; }; // NOLINT
05925
```

```
05926
               template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<528>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<857, 5> { using ZPZ = aerobus::zpz<857>; using type =
05927
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<854»; }; // NOLINT
         template<> struct ConwayPolynomial<857, 6> { using ZPZ = aerobus::zpz<857>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<824>, ZPZV<65>, ZPZV<3»; }; // NOLINT</pre>
05928
               template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::2pz<857>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5
              template<> struct ConwayPolynomial<857, 8> { using ZPZ = aerobus::zpz<857>; using type =
05930
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<611>, ZPZV<552>, ZPZV<494>, ZPZV<3»; }; //
         NOLINT
              template<> struct ConwayPolynomial<857, 9> { using ZPZ = aerobus::zpz<857>; using type =
05931
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<719>, ZPZV<854»;
         }; // NOLINT
05932
               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 =
05933
         POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type =
05935
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<859, 5> { using ZPZ = aerobus::zpz<859>; using type =
05936
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<857»; }; // NOLINT
05937
               template<> struct ConwayPolynomial<859, 6> { using ZPZ = aerobus::zpz<859>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<419>, ZPZV<646>, ZPZV<566>, ZPZV<566>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpz<859>; using type =
05938
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 8> { using ZPZ = aerobus::zpz<859>; using type =
05939
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<52>, ZPZV<446>, ZPZV<672>, ZPZV<672>; }; //
         NOLINT
05940
               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
05941
               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 =
05942
         POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; }; // NOLINT
05943
               template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<588%; }; // NOLINT template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
05944
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<863, 5> { using ZPZ = aerobus::zpz<863>; using type =
05945
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<858»; }; // NOLINT
               template<> struct ConwayPolynomial<863, 6> { using ZPZ = aerobus::zpz<863>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<330>, ZPZV<62>, ZPZV<300>, ZPZV<5»; };
05947
              template<> struct ConwayPolynomial<863, 7> { using ZPZ = aerobus::zpz<863>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<858»; }; // NOLINT
05948
              template<> struct ConwayPolynomial<863, 8> { using ZPZ = aerobus::zpz<863>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<765>, ZPZV<576>, ZPZV<849>, ZPZV<5»; }; //
               template<> struct ConwayPolynomial<863, 9> { using ZPZ = aerobus::zpz<863>; using type =
05949
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<381>, ZPZV<381, ZPZV<45, ZPZV<858»; };
         // NOLINT
              template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
05950
         POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
         POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
05952
        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 =
05953
05954
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<875»; }; // NOLINT
05955
              template<> struct ConwayPolynomial<877, 6> { using ZPZ = aerobus::zpz<877>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<400>, ZPZV<855>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<877, 7> { using ZPZ = aerobus::zpz<877>; using type =
05956
         POLYV<ZPZV<1>, ZPZV<0>, 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 =
05957
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<767>, ZPZV<319>, ZPZV<347>, ZPZV<2»; };
05958
              template<> struct ConwayPolynomial<877, 9> { using ZPZ = aerobus::zpz<877>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<770>, ZPZV<778>, ZPZV<875»;
         }; // NOLINT
05959
               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 =
05960
         POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT
05961
               template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<878»; }; // NOLINT template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
05962
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<447>, ZPZV<3»; };
                                                                                             // NOLINT
               template<> struct ConwayPolynomial<881, 5> { using ZPZ = aerobus::zpz<881>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<878»; }; // NOLINT
05964
              template<> struct ConwayPolynomial<881, 6> { using ZPZ = aerobus::zpz<881>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<419>, ZPZV<231>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<881, 7> { using ZPZ = aerobus::zpz<881>; using type =
05965
```

```
POLYV<ZPZV<1>, 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<561>, ZPZV<3»; }; //
         NOLINT
05967
               template<> struct ConwayPolynomial<881, 9> { using ZPZ = aerobus::zpz<881>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<587>, ZPZV<510>, ZPZV<878»;
         }; // NOLINT
05968
               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 =
05969
        POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<883, 3> { using ZPZ = aerobus::zpz<883>; using type =
05970
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<6>, ZPZV<681»; }; // NOLINT template<> struct ConwayPolynomial<883, 4> { using ZPZ = aerobus::zpz<883>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2»; }; // NOLINT
05972
               template<> struct ConwayPolynomial<883, 5> { using ZPZ = aerobus::zpz<883>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<881»; }; // NOLINT
        template<> struct ConwayPolynomial<883, 6> { using ZPZ = aerobus::zpz<883>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<879>, ZPZV<865>, ZPZV<871>, ZPZV<2»; }; // NOLINT
05973
05974
               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»; }; //
05975
               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<29; }; //
         NOLINT
05976
               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
05977
               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 =
05978
        POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; }; // NOLINT
05979
               template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<882»; }; // NOLINT template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
05980
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<887, 5> { using ZPZ = aerobus::zpz<887>; using type =
05981
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<5>, ZPZY<882»; }; // NOLINT template<> struct ConwayPolynomial<887, 6> { using ZPZ = aerobus::zpz<887>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<775>, ZPZV<341>, ZPZV<28>, ZPZV<5»; };
              template<> struct ConwayPolynomial<887, 7> { using ZPZ = aerobus::zpz<887>; using type =
05983
        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 =
05984
         POLYV<7P7V<1>. 7P7V<0>. 7P7V<0>. 7P7V<0>. 7P7V<0>. 7P7V<5>: }: //
               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
05986
               template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
        POLYV<ZPZV<1>, ZPZV<905»; }; // NOLINT
               template<> struct ConwayPolynomial<907, 2> { using ZPZ = aerobus::zpz<907>; using type =
05987
        POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2»; }; // NOLINT
               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 =
05989
        POLYV<ZPZV<1>, ZPZV<1>, ZPZV<14>, ZPZV<478>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
05990
         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 =
05991
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<626>, ZPZV<266>, ZPZV<266>, ZPZV<266>, ZPZV<20,; }; // NOLINT template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type =
05992
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<905»; }; // NOLINT template<> struct ConwayPolynomial<907, 8> { using ZPZ = aerobus::zpz<907>; using type =
05993
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<584>, ZPZV<518>, ZPZV<811>, ZPZV<2»; }; //
05994
              template<> struct ConwayPolynomial<907, 9> { using ZPZ = aerobus::zpz<907>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<783>, ZPZV<57>, ZPZV<905»;
         }; // NOLINT
05995
               template<> struct ConwayPolynomial<911, 1> { using ZPZ = aerobus::zpz<911>; using type =
        POLYV<ZPZV<1>, ZPZV<894»; }; // NOLINT
05996
               template<> struct ConwayPolynomial<911, 2> { using ZPZ = aerobus::zpz<911>; using type =
         POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; };
                                                                     // NOLINT
05997
              template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
               'V<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; }; // NOLINT
template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; };
05998
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17*; }; // NOLINT template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<894»; }; // NOLINT
06000
               template<> struct ConwayPolynomial<911, 6> { using ZPZ = aerobus::zpz<911>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<683>, ZPZV<19>, ZPZV<17, }; // NOLINT template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
06001
         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
06002
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<590>, ZPZV<168>, ZPZV<17»; };
         NOLINT
06003
              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<679>, ZPZV<116>, ZPZV<894»;
         }; // NOLINT
```

```
06004
               template<> struct ConwayPolynomial<919, 1> { using ZPZ = aerobus::zpz<919>; using type =
         POLYV<ZPZV<1>, ZPZV<912»; }; // NOLINT
06005
              template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
         POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; }; // NOLINT
06006
               template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912*; }; // NOLINT template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7»; }; // NOLINT
              template<> struct ConwayPolynomial<919, 5> { using ZPZ = aerobus::zpz<919>; using type =
06008
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<912»; }; // NOLINT
06009
               template<> struct ConwayPolynomial<919, 6> { using ZPZ = aerobus::zpz<919>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<312>, ZPZV<817>, ZPZV<113>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<919, 7> { using ZPZ = aerobus::zpz<919>; using type
06010
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<912»; }; //
06011
              template<> struct ConwayPolynomial<919, 8> { using ZPZ = aerobus::zpz<919>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<202>, ZPZV<504>, ZPZV<50; }; //
         NOLTNT
         template<> struct ConwayPolynomial<919, 9> { using ZPZ = aerobus::zpz<919>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<410>, ZPZV<623>, ZPZV<912»;
06012
               template<> struct ConwayPolynomial<929, 1> { using ZPZ = aerobus::zpz<929>; using type =
06013
         POLYV<ZPZV<1>, ZPZV<926»; }; // NOLINT
               template<> struct ConwayPolynomial<929, 2> { using ZPZ = aerobus::zpz<929>; using type =
06014
         POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
06015
         POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<5
06016
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3»; }; // NOLINT
06017
               template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<92; }; // NOLINT template<> struct ConwayPolynomial<929, 6> { using ZPZ = aerobus::zpz<929>; using type =
06018
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<805>, ZPZV<92>, ZPZV<86>, ZPZV<3»; };
                                                                                                                            // NOLINT
               template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type =
06019
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<926»; }; // NOLINT
06020
              template<> struct ConwayPolynomial<929, 8> { using ZPZ = aerobus::zpz<929>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699>, ZPZV<292>, ZPZV<586>, ZPZV<3»; }; //
         NOLINT
               template<> struct ConwayPolynomial<929, 9> { using ZPZ = aerobus::zpz<929>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<481>, ZPZV<199>, ZPZV<926»;
         }; // NOLINT
06022
               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 =
06023
         POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<937, 3> { using ZPZ = aerobus::zpz<937>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<932»; };
                                                                                // NOLINT
06025
              template<> struct ConwayPolynomial<937, 4> { using ZPZ = aerobus::zpz<937>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<23>, ZPZV<585>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
06026
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<932»; }; // NOLINT
06027
               template<> struct ConwayPolynomial<937, 6> { using ZPZ = aerobus::zpz<937>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<794>, ZPZV<727>, ZPZV<934>, ZPZV<5»; }; // NOLINT
06028
              template<> struct ConwayPolynomial<937, 7> { using ZPZ = aerobus::zpz<937>; using type =
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<932»; }; // NoLI template<> struct ConwayPolynomial<937, 8> { using ZPZ = aerobus::zpz<937>; using type =
                                                                                                                                        // NOJ.TNT
06029
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<26>, ZPZV<53>, ZPZV<5»; };
06030
              template<> struct ConwayPolynomial<937, 9> { using ZPZ = aerobus::zpz<937>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<533>, ZPZV<483>, ZPZV<932»;
         }; // NOLINT
06031
               template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
         POLYV<ZPZV<1>, ZPZV<939»; }; // NOLINT
06032
               template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
         POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2»; }; // NOLINT
06033
              template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3939»; }; // NOLINT
template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<941, 5> { using ZPZ = aerobus::zpz<941>; using type =
06034
06035
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<939»; }; // NOLINT
               template<> struct ConwayPolynomial<941, 6> { using ZPZ = aerobus::zpz<941>; using type =
06036
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<459>, ZPZV<694>, ZPZV<538>, ZPZV<2»; }; // NOLINT
06037
               template<> struct ConwayPolynomial<941, 7> { using ZPZ = aerobus::zpz<941>; using type =
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 8> { using ZPZ = aerobus::zpz<941>; using type =
06038
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<80>, ZPZV<675>, ZPZV<590>, ZPZV<2»; }; //
         NOLINT
06039
              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
06040
               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 =
         POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
06042
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 4> { using ZPZ = aerobus::zpz<947>; using type =
```

9.3 aerobus.h 205

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2»; };
                 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
06045
                 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
06046
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 >, ZP
06047
                 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»; }; //
          template<> struct ConwayPolynomial<947, 9> { using ZPZ = aerobus::zpz<947>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<808>, ZPZV<945»;
06048
          }; // NOLINT
  template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<950»; }; // NOLINT
06050
                template<> struct ConwayPolynomial<953, 2> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3»; }; // NOLINT
                 template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
06051
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950»; }; // NOLINT
                 template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<865>, ZPZV<3»; }; // NOLINT
06053
                 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</pre>
06054
                 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<950»; };
                template<> struct ConwayPolynomial<953, 8> { using ZPZ = aerobus::zpz<953>; using type =
06056
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<579>, ZPZV<658>, ZPZV<108>, ZPZV<3»; }; //
          NOLINT
                template<> struct ConwayPolynomial<953, 9> { using ZPZ = aerobus::zpz<953>; using type =
06057
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<819>, ZPZV<316>, ZPZV<950»;
          }; // NOLINT
                 template<> struct ConwayPolynomial<967, 1> { using ZPZ = aerobus::zpz<967>; using type =
06058
          POLYV<ZPZV<1>, ZPZV<962»; }; // NOLINT
                 template<> struct ConwayPolynomial<967, 2> { using ZPZ = aerobus::zpz<967>; using type =
06059
          POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5»; }; // NOLINT
                 template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
06061
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; }; // NOLINT
          template<> struct ConwayPolynomial<967, 5> { using ZPZ = aerobus::zpz<967>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<962»; }; // NOLINT
06062
06063
                 template<> struct ConwayPolynomial<967, 6> { using ZPZ = aerobus::zpz<967>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<948>, ZPZV<831>, ZPZV<5»; }; // NOLINT
06064
                 template<> struct ConwayPolynomial<967, 7> { using ZPZ = aerobus::zpz<967>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 8> { using ZPZ = aerobus::zpz<967>; using type =
06065
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<840>, ZPZV<502>, ZPZV<1365, ZPZV<5»; }; //
          NOLINT
06066
                 template<> struct ConwayPolynomial<967, 9> { using ZPZ = aerobus::zpz<967>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<512>, ZPZV<783>, ZPZV<962»;
          }; // NOLINT
06067
                 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 =
06068
          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 =
06070
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<527>, ZPZV<527>, ZPZV<6»; }; // NOLINT
template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
06071
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<965»; }; // NOLINT
                 template<> struct ConwayPolynomial<971, 6> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<2PZV<1>, 2PZV<0>, 2PZV<1>, 2PZV<970>, 2PZV<729>, 2PZV<718>, 2PZV<6»; }; // NOLINT
06073
                template<> struct ConwayPolynomial<971, 7> { using ZPZ = aerobus::zpz<971>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<965»; }; // NOLINT template<> struct ConwayPolynomial<971, 8> { using ZPZ = aerobus::zpz<971>; using type =
06074
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28+, ZPZV<281, ZPZV<206>, ZPZV<6»; }; //
          NOLINT
                 template<> struct ConwayPolynomial<971, 9> { using ZPZ = aerobus::zpz<971>; using type =
06075
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<473>, ZPZV<965»;
          }; // NOLINT
06076
                 template<> struct ConwayPolynomial<977, 1> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<974»; }; // NOLINT
                 template<> struct ConwayPolynomial<977, 2> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; }; // NOLINT
06078
                 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 =
06079
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<800>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<977, 5> { using ZPZ = aerobus::zpz<977>; using type =
06080
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<974»; }; // NOLINT
06081
                template<> struct ConwayPolynomial<977, 6> { using ZPZ = aerobus::zpz<977>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<729>, ZPZV<830>, ZPZV<753>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<977, 7> { using ZPZ = aerobus::zpz<977>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7
06082
```

206 File Documentation

```
template<> struct ConwayPolynomial<977, 8> { using ZPZ = aerobus::zpz<977>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<855>, ZPZV<807>, ZPZV<77>, ZPZV<3»; };
      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<450>, ZPZV<450>, ZPZV<740>, ZPZV<740>,
06084
       }; // NOLINT
           template<> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
       POLYV<ZPZV<1>, ZPZV<978»; }; // NOLINT
           template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
06086
      POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
06087
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978»; }; // NOLINT template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
06088
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5»; }; // NOLINT
06089
          template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
      06090
           template<> struct ConwayPolynomial<983, 6> { using ZPZ = aerobus::zpz<983>; using type =
      POLYVCZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<849>, ZPZV<265, ZPZV<288, ZPZV<288, ZPZV<2883; is // NOLINT template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<978»; };
           template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<5»; }; //
      template<> struct ConwayPolynomial<983, 9> { using ZPZ = aerobus::zpz<983>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<85*>, ZPZV<85*>, ZPZV<87>, ZPZV<978*;</pre>
06093
       }; // NOLINT
06094
           template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<985»; }; // NOLINT
           template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
06095
      POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; }; // NOLINT
           template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
06096
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985»; }; // NOLINT
           template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
06097
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6»; }; // NOLINT
06098
           template<> struct ConwayPolynomial<991, 5> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<985»; }; // NOLINT
      template<> struct ConwayPolynomial<991, 6> { using ZPZ = aerobus::zpz<991>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<637>, ZPZV<855>, ZPZV<278>, ZPZV<6»; }; // NOLINT
06099
06100
           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<0>, ZPZV<7>, ZPZV<985»; };
06101
          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»; }; //
       NOLINT
06102
           template<> struct ConwayPolynomial<991, 9> { using ZPZ = aerobus::zpz<991>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<466>, ZPZV<222>, ZPZV<985»;
       }; // NOLINT
06103
           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 =
06104
      POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
06105
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990»; }; // NOLINT
           template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
06106
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
06107
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<990»; }; // NOLINT template<> struct ConwayPolynomial<997, 6> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<981>, ZPZV<58>, ZPZV<260>, ZPZV<7»; }; // NOLINI
           template<> struct ConwayPolynomial<997, 7> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<990»; }; // NOLIN template<> struct ConwayPolynomial<997, 8> { using ZPZ = aerobus::zpz<997>; using type =
06110
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<473>, ZPZV<473>, ZPZV<241>, ZPZV<7»; }; //
      NOLINT
           template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<732>, ZPZV<616>, ZPZV<990»;
       }; // NOLINT
06112 #endif // DO NOT DOCUMENT
06113 } // namespace aerobus
06114 #endif // AEROBUS_CONWAY_IMPORTS
06116 #endif // __INC_AEROBUS__ // NOLINT
```

9.4 src/examples.h File Reference

9.5 examples.h

Go to the documentation of this file.

```
00001 #ifndef SRC_EXAMPLES_H_
00002 #define SRC_EXAMPLES_H_
00050 #endif // SRC_EXAMPLES_H_
```

Chapter 10

Examples

10.1 examples/hermite.cpp

How to use aerobus::known_polynomials::hermite_phys polynomials

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

10.2 examples/custom_taylor.cpp

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

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

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

template<size_t deg>
```

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

10.3 examples/fp16.cu

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

```
// TO compile with nvcc -O3 -std=c++20 -arch=sm_90 fp16.cu
// \ {\tt TO GET} \ {\tt optimal performances, modify cuda\_fpl6.h \ by adding $\_$CUDA\_FPl6\_CONSTEXPR$\_$ to line 5039 (version of the context of
               12.6)
#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\ \mathrm{bits}
using int_type = aerobus::i16;
using float_type = __half2;
constexpr size t N = 1 « 24;
template<typename T>
struct ExpmlDegree;
template<>
struct Expm1Degree<double> {
          static constexpr size_t val = 18;
template<>
struct Expm1Degree<float> {
           static constexpr size_t val = 11;
template<>
struct Expm1Degree<__half2> {
         static constexpr size_t val = 6;
template<>
struct ExpmlDegree<__half> {
         static constexpr size_t val = 6;
double rand(double min, double max) {
    double range = (max - min);
double div = RAND_MAX / range;
    return min + (rand() / div); // NOLINT
template<tvpename T>
struct GetRandT;
struct GetRandT<double> {
          static double func(double min, double max) {
                    return rand(min, max);
};
template<>
struct GetRandT<float> {
          static float func(double min, double max) {
   return (float) rand(min, max);
};
template<>
struct GetRandT<__half2> {
    static __half2 func(double min, double max) {
                    return __half2(__float2half((float)rand(min, max)), __float2half((float)rand(min, max)));
```

```
};
template<>
struct GetRandT<__half> {
    static __half func(double min, double max) {
         return __float2half((float)rand(min, max));
using EXPM1 = aerobus::expm1<int_type, Expm1Degree<float_type>::val>;
__device__ INLINED float_type f(float_type x) {
    return EXPM1::eval(x);
__global__ void run(size_t N, float_type* in, float_type* out) {
   for(size_t i = threadIdx.x + blockDim.x * blockIdx.x; i < N; i += blockDim.x * gridDim.x) {</pre>
         out[i] = f(f(f(f(f(f(f(f(f(f(f(in[i]))))))))));
int main() {
     float_type *d_in, *d_out;
     cudaMalloc<float_type>(&d_in, N * sizeof(float_type));
     cudaMalloc<float_type>(&d_out, N * sizeof(float_type));
     float_type *out = reinterpret_cast<float_type*>(malloc(N * sizeof(float_type)));
     for(size_t i = 0; i < N; ++i) {
         in[i] = GetRandT<float_type>::func(-0.01, 0.01);
     cudaMemcpy(d_in, in, N * sizeof(float_type), cudaMemcpyHostToDevice);
     run«<128, 512»>(N, d in, d out);
    cudaMemcpy(out, d_out, N * sizeof(float_type), cudaMemcpyDeviceToHost);
     cudaFree(d in);
     cudaFree(d out);
// generated SASS :
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875; HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125; HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625;
HFMA2.MMA R5, R6, R5, 0.5, 0.5;
HFMA2 R5, R6, R5, 1, 1;
HFMA2.MMA R5, R6, R5, RZ;
HFMA2.RMA R7, R5, R2.H0_H0, 0.0013885498046875, 0.0013885498046875; HFMA2.MMA R7, R5, R7, 0.008331298828125, 0.008331298828125; HFMA2 R7, R5, R7, 0.041656494140625, 0.041656494140625;
HFMA2.MMA R7, R5, R7, 0.1666259765625, 0.1666259765625;
HFMA2.MMA R7, R5, R7, 0.5, 0.5;

HFMA2.MMA R7, R5, R7, 1, 1;

HFMA2 R7, R5, R7, RZ.H0_H0;

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

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

HFMA2 R5, R6, R5, 1, 1;

HFMA2.MMA R6, R6, R5, RZ;

HFMA2 R5, R6, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875;
HFMA2.MMA R5, R6, R5, 0.008331298828125, 0.008331298828125;
HFMA2 R5, R6, R5, 0.041656494140625, 0.041656494140625;
HFMA2.MMA R5, R6, R5, 0.1666259765625, 0.1666259765625;
HFMA2 R5, R6, R5, 0.5, 0.5;
HFMA2.MMA R7, R6, R5, 1, 1;
IADD3.X R5, R8, UR11, RZ, P0, !PT;
IADD3 R3, P0, R2, R3, RZ;
IADD3.X RO, RZ, RO, RZ, PO, !PT;
ISETP.GE.U32.AND PO, PT, R3, UR8, PT;
HFMA2 R7, R6, R7, RZ.H0_H0;
ISETP.GE.U32.AND.EX P0, PT, R0, UR9, PT, P0;
STG.E desc[UR6][R4.64], R7;
```

10.4 examples/continued_fractions.cpp

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

10.5 examples/modular_arithmetic.cpp

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

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

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

// x^3 + 2x^2 + 1, with coefficients in Z/2Z, actually x^3 + 1
using P = aerobus::make_int_polynomial_t<FIELD, 1, 2, 0, 1>;
// x^3 + 5x^2 + 7x + 11 with coefficients in Z/17Z, meaning actually x^3 + x^2 + 1
using Q = aerobus::make_int_polynomial_t<FIELD, 1, 5, 8, 1>;

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

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

10.6 examples/make_polynomial.cpp

```
How to build your own sequence of known polynomials, here Abel polynomials
#include <iostream
#include "../src/aerobus.h"
// let's build Abel polynomials from scratch using Aerobus
// note : it's now integrated in the main library, but still serves as an example
template<typename I = aerobus::i64>
struct AbelHelper {
 private:
    using P = aerobus::polynomial<I>;
    // to keep recursion working, we need to operate on a\!*\!n and not just a
    template<size_t deg, I::inner_type an>
    struct Inner {
   // abel(n, a) = (x-an) * abel(n-1, a)
   using type = typename aerobus::mul_t
             typename Inner<deg-1, an>::type,
             typename aerobus::sub_t<typename P::X, typename P::template inject_constant_t<an>>
    };
    // abel(0, a) = 1
    template<I::inner_type an>
    struct Inner<0, an> {
         using type = P::one;
    // abel(1, a) = X
template<I::inner_type an>
    struct Inner<1, an>
        using type = P::X;
};
template<size_t n, auto a, typename I = aerobus::i64>
using AbelPolynomials = typename AbelHelper<I>::template Inner<n, a*n>::type;
using A2_3 = AbelPolynomials<3, 2>;
int main() {
    std::cout « "expected = x^3 - 12 x^2 + 36 x" « std::endl;
std::cout « "aerobus = " « A2_3::to_string() « std::endl;
    return 0;
```

10.7 examples/polynomials_over_finite_field.cpp

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

```
#include <iostream>
```

```
#include "../src/aerobus.h"

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

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

10.8 examples/compensated_horner.cpp

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

See also

```
publication
// run with ./generate_comp_horner.sh in this directory
// that will compile and run this sample and plot all the generated data
#include "../src/aerobus.h"
using namespace aerobus; // NOLINT
constexpr size t NB POINTS = 400;
template<typename P, typename T, bool compensated>
DEVICE INLINED T eval(const T& x) {
    if constexpr (compensated) {
        return P::template compensated_eval<T>(x);
    } else {
        return P::template eval<T>(x);
template<typename T>
DEVICE T exact_large(const T& x) {
    return pow_scalar<T, 5>(0.75 - x) * pow_scalar<T, 11>(1 - x);
template<typename T>
DEVICE T exact_small(const T& x) {
    return pow_scalar<T, 3>(x - 1);
template<typename P, typename T, bool compensated>
void run(T left, T right, const char *file_name, T (*exact)(const T&)) {
    FILE *f = ::fopen(file_name, "w+");
T step = (right - left) / NB_POINTS;
    T x = left;
    for (size_t i = 0; i <= NB_POINTS; ++i) {</pre>
         ::fprintf(f, "%e %e %e\n", x, eval<P, T, compensated>(x), exact(x));
    ::fclose(f);
}
int main() {
         // (0.75 - x)^5 * (1 - x)^11
         using P = mul_t <
             pow_t<pq64, pq64::val<</pre>
                  typename q64::template inject_constant_t<-1>,
                  q64::val<i64::val<3>, i64::val<4>», 5>,
             pow_t<pq64, pq64::val<typename q64::template inject_constant_t<-1>, typename q64::one>, 11>
         using FLOAT = double;
         run<P, FLOAT, false>(0.68, 1.15, "plots/large_sample_horner.dat", &exact_large);
run<P, FLOAT, true>(0.68, 1.15, "plots/large_sample_comp_horner.dat", &exact_large);
         run<P, FLOAT, false>(0.74995, 0.75005, "plots/first_root_horner.dat", &exact_large);
```

```
run<P, FLOAT, true>(0.74995, 0.75005, "plots/first_root_comp_horner.dat", &exact_large);
run<P, FLOAT, false>(0.9935, 1.0065, "plots/second_root_horner.dat", &exact_large);
run<P, FLOAT, true>(0.9935, 1.0065, "plots/second_root_comp_horner.dat", &exact_large);
}

// (x - 1) ^ 3
using P = make_int_polynomial_t<i32, 1, -3, 3, -1>;

run<P, double, false>(1-0.00005, 1+0.00005, "plots/double.dat", &exact_small);
run<P, float, true>(1-0.00005, 1+0.00005, "plots/float_comp.dat", &exact_small);
}
```

Index

```
abs t
                                                              mulfractions t, 29
     aerobus, 20
                                                              pi64, 30
add t
                                                              PI fraction, 30
    aerobus, 20
                                                              pow t, 30
    aerobus::i32, 59
                                                              pq64, 30
    aerobus::i64, 65
                                                              q32, 30
    aerobus::polynomial < Ring >, 74
                                                              q64, 31
    aerobus::Quotient < Ring, X >, 82
                                                              sin, 31
    aerobus::zpz , 107
                                                              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 >, 47
    atan, 21
    atanh, 21
                                                              type, 47
    bell t, 23
                                                              val, 48
    bernoulli t, 23
                                                         aerobus::ContinuedFraction < a0, rest... >, 48
    bernoulli v, 35
                                                              type, 49
    combination t, 23
                                                              val, 49
    combination v, 35
                                                         aerobus::ContinuedFraction < values >, 46
                                                         aerobus::ConwayPolynomial, 49
    cos, 23
    cosh, 25
                                                         aerobus::Embed< i32, i64 >, 51
    div t, 25
                                                              type, 51
    E fraction, 25
                                                         aerobus::Embed< polynomial< Small >, polynomial<
    embed_int_poly_in_fractions_t, 25
                                                                  Large >>, 52
    exp, 26
                                                              type, 52
    expm1, 26
                                                         aerobus::Embed < q32, q64 >, 53
    factorial t, 26
                                                              type, 53
                                                         aerobus::Embed< Quotient< Ring, X >, Ring >, 54
    factorial_v, 35
    field, 34
                                                              type, 54
    fpq32, 26
                                                         aerobus::Embed< Ring, FractionField< Ring >>, 55
    fpq64, 27
                                                              type, 55
                                                         aerobus::Embed< Small, Large, E >, 51
    FractionField, 27
    gcd_t, 27
                                                         aerobus::Embed< zpz< x>, i32>, 55
    geometric sum, 27
                                                              type, 56
    Inp1, 27
                                                         aerobus::i32, 57
                                                              add t, 59
    make_frac_polynomial_t, 28
    make int polynomial t, 28
                                                              div t, 59
    make q32 t, 28
                                                              eq t, 59
    make_q64_t, 29
                                                              eq_v, 62
    makefraction_t, 29
                                                              gcd_t, 59
    mul t, 29
                                                              gt_t, 60
```

inject_constant_t, 60	aerobus::known_polynomials, 40
inject_ring_t, 60	hermite_kind, 40
inner_type, 60	physicist, 41
is_euclidean_domain, 62	probabilist, 41
is_field, 62	aerobus::polynomial < Ring >, 72
It t, 60	add_t, 74
mod_t, 61	derive_t, 74
mul_t, 61	div_t, 74
one, 61	eq_t, 75
pos_t, 61	gcd_t, 75
pos_v, 62	gt_t, 75
sub_t, 62	inject_constant_t, 76
zero, 62	inject_ring_t, 76
aerobus::i32::val< x >, 91	is_euclidean_domain, 80
enclosing_type, 92	is_field, 80
get, 92	lt_t, 76
is_zero_t, 92	mod_t, 76
to_string, 92	monomial_t, 77
v, 92	mul_t, 77
aerobus::i64, 64	one, 77
add_t, 65	pos_t, 77
div_t, 66	pos_v, 80
eq_t, 66	simplify_t, 79
eq_v, 69	sub_t, 79
gcd_t, 66	X, 79
gt_t, 66	zero, 79
gt_v, 69	aerobus::polynomial< Ring >::compensated_horner<
inject_constant_t, 67	arithmeticType, P >::EFTHorner< index,
inject_ring_t, 67	ghost $>$, 49
inner_type, 67	func, 50
is_euclidean_domain, 69	aerobus::polynomial< Ring >::compensated_horner<
is_field, 69	arithmeticType, P >::EFTHorner<-1, ghost >,
lt_t, 67	50
lt_v, 70	func, 50
mod_t, 68	aerobus::polynomial< Ring >::horner_reduction_t< P
mul_t, 68	>, 56
one, 68	aerobus::polynomial< Ring >::horner_reduction_t< P
pos_t, 68	>::inner< index, stop >, 70
pos_v, 70	type, 71
sub_t, 68	aerobus::polynomial< Ring >::horner_reduction_t< P
zero, 69	>::inner< stop, stop >, 71
aerobus::i64::val $< x >$, 93	type, 71
enclosing_type, 94	aerobus::polynomial< Ring >::val< coeffN >, 102
get, 94	aN, 103
inner_type, 94	coeff_at_t, 103
is_zero_t, 94	compensated_eval, 104
to_string, 94	degree, 105
v, 95	enclosing_type, 103
aerobus::internal, 36	eval, 104
index_sequence_reverse, 40	is_zero_t, 103
is_instantiation_of_v, 40	is_zero_v, 105
make_index_sequence_reverse, 40	ring_type, 104
type_at_t, 40	strip, 104
aerobus::is_prime< n >, 71	to_string, 104
value, 72	value_at_t, 104
aerobus::IsEuclideanDomain, 43	aerobus::polynomial < Ring >::val < coeffN >::coeff_at <
aerobus::IsField, 43	index, E >, 45
aerobus::IsRing, 44	aerobus::polynomial< Ring >::val< coeffN >::coeff_at<

```
index, std::enable_if_t<(index< 0 | | index >
                                                               add_t, 107
                                                               div t, 107
          0)>>, 45
     type, 45
                                                               eq_t, 108
aerobus::polynomial < Ring >::val < coeffN >::coeff_at <
                                                               eq_v, 111
          index, std::enable_if_t<(index==0)>>, 46
                                                               gcd_t, 108
                                                               gt t, 108
aerobus::polynomial< Ring >::val< coeffN, coeffs >,
                                                               gt_v, 111
          95
                                                               inject_constant_t, 109
     aN, 96
                                                               inner type, 109
     coeff_at_t, 96
                                                               is euclidean domain, 111
     compensated_eval, 97
                                                               is field, 111
     degree, 99
                                                               It_t, 109
     enclosing_type, 96
                                                               lt_v, 111
     eval, 98
                                                               mod t, 109
                                                               mul_t, 109
     is_zero_t, 97
     is_zero_v, 99
                                                               one, 110
     ring type, 97
                                                               pos t, 110
     strip, 97
                                                               pos v, 112
     to_string, 98
                                                               sub_t, 110
     value_at_t, 97
                                                               zero, 110
aerobus::Quotient < Ring, X >, 81
                                                          aerobus::zpz<p>::val<math><x>, 100
     add t, 82
                                                               enclosing_type, 101
     div_t, 83
                                                               get, 101
     eq_t, 83
                                                               is_zero_t, 101
     eq_v, 85
                                                               is zero v, 102
     inject_constant_t, 83
                                                               to_string, 101
     inject_ring_t, 83
                                                               v, 102
     is euclidean domain, 85
                                                          aligned malloc
     mod t, 84
                                                               aerobus, 34
     mul t, 84
                                                          alternate t
     one, 84
                                                               aerobus, 20
     pos_t, 84
                                                          alternate_v
     pos v, 85
                                                               aerobus, 35
                                                          aN
     zero, 85
aerobus::Quotient < Ring, X >::val < V >, 99
                                                               aerobus::polynomial < Ring >::val < coeffN >, 103
     raw_t, 100
                                                               aerobus::polynomial< Ring >::val< coeffN, coeffs
     type, 100
                                                                    >, 96
                                                          asin
aerobus::type_list< Ts >, 87
     at, 88
                                                               aerobus, 21
     concat, 88
                                                          asinh
     insert, 88
                                                               aerobus, 21
     length, 89
                                                          at
     push back, 88
                                                               aerobus::type_list< Ts >, 88
     push front, 89
                                                          atan
     remove, 89
                                                               aerobus, 21
aerobus::type_list< Ts >::pop_front, 80
                                                          atanh
     tail, 81
                                                               aerobus, 21
     type, 81
                                                          bell t
aerobus::type_list< Ts >::split< index >, 86
                                                               aerobus, 23
     head, 86
                                                          bernoulli t
     tail, 86
                                                               aerobus, 23
aerobus::type list<>, 90
                                                          bernoulli v
     concat, 90
                                                               aerobus, 35
     insert, 90
     length, 91
                                                          coeff at t
     push_back, 90
                                                               aerobus::polynomial < Ring >::val < coeffN >, 103
     push_front, 90
                                                               aerobus::polynomial< Ring >::val< coeffN, coeffs
aerobus::zpz , 105
                                                                    >, <mark>96</mark>
```

```
combination_t
                                                            aerobus, 26
     aerobus, 23
                                                       factorial_t
combination v
                                                            aerobus, 26
    aerobus, 35
                                                       factorial v
compensated eval
                                                            aerobus, 35
     aerobus::polynomial < Ring >::val < coeffN >, 104
                                                       field
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                            aerobus, 34
          >, 97
                                                       fpq32
concat
                                                            aerobus, 26
     aerobus::type_list< Ts >, 88
                                                       fpq64
     aerobus::type_list<>, 90
                                                            aerobus, 27
cos
                                                       FractionField
     aerobus, 23
                                                            aerobus, 27
cosh
     aerobus, 25
                                                       func
                                                            aerobus::polynomial < Ring >::compensated horner <
                                                                 arithmeticType, P >::EFTHorner< index,
degree
     aerobus::polynomial < Ring >::val < coeffN >, 105
                                                                 ghost >, 50
                                                            aerobus::polynomial < Ring >::compensated_horner <
     aerobus::polynomial < Ring >::val < coeffN, coeffs
                                                                 arithmeticType, P >::EFTHorner <- 1, ghost >,
derive t
     aerobus::polynomial < Ring >, 74
                                                       gcd t
div t
                                                            aerobus, 27
     aerobus, 25
                                                            aerobus::i32, 59
     aerobus::i32, 59
                                                            aerobus::i64, 66
     aerobus::i64, 66
                                                            aerobus::polynomial< Ring >, 75
     aerobus::polynomial < Ring >, 74
                                                            aerobus::zpz< p>, 108
     aerobus::Quotient < Ring, X >, 83
                                                        geometric_sum
     aerobus::zpz, 107
                                                            aerobus, 27
E fraction
                                                       get
     aerobus, 25
                                                            aerobus::i32::val< x >, 92
embed_int_poly_in_fractions_t
                                                            aerobus::i64::val < x > , 94
    aerobus, 25
                                                            aerobus::zpz ::val < x >, 101
enclosing type
                                                       gt_t
     aerobus::i32::val< x >, 92
                                                            aerobus::i32, 60
     aerobus::i64::val < x >, 94
                                                            aerobus::i64, 66
     aerobus::polynomial < Ring >::val < coeffN >, 103
                                                            aerobus::polynomial < Ring >, 75
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                            aerobus::zpz , 108
          >, 96
                                                       gt_v
     aerobus::zpz ::val < x >, 101
                                                            aerobus::i64, 69
eq t
                                                            aerobus::zpz, 111
     aerobus::i32, 59
                                                       head
     aerobus::i64, 66
                                                            aerobus::type list< Ts >::split< index >, 86
     aerobus::polynomial < Ring >, 75
     aerobus::Quotient < Ring, X >, 83
                                                       hermite kind
                                                            aerobus::known_polynomials, 40
     aerobus::zpz, 108
eq_v
                                                       index_sequence_reverse
     aerobus::i32, 62
                                                            aerobus::internal, 40
     aerobus::i64, 69
                                                       inject constant t
     aerobus::Quotient< Ring, X >, 85
                                                            aerobus::i32, 60
     aerobus::zpz, 111
                                                            aerobus::i64, 67
eval
                                                            aerobus::polynomial < Ring >, 76
     aerobus::polynomial < Ring >::val < coeffN >, 104
                                                            aerobus::Quotient < Ring, X >, 83
     aerobus::polynomial< Ring >::val< coeffN, coeffs
                                                            aerobus::zpz , 109
         >, 98
                                                       inject_ring_t
exp
                                                            aerobus::i32, 60
    aerobus, 26
                                                            aerobus::i64, 67
expm1
```

aerobus::polynomial < Ring >, 76	make_q64_t
aerobus::Quotient $<$ Ring, $X>$, 83	aerobus, 29
inner_type	makefraction_t
aerobus::i32, 60	aerobus, 29
aerobus::i64, 67	mod_t
aerobus::i64::val $< x >$, 94	aerobus::i32, 61
aerobus::zpz, 109	aerobus::i64, 68
insert	aerobus::polynomial $<$ Ring $>$, 76
aerobus::type_list< Ts >, 88	aerobus::Quotient $<$ Ring, X $>$, 84
aerobus::type_list<>, 90	aerobus:: $zpz $, 109
Introduction, 1	monomial_t
is_euclidean_domain	aerobus::polynomial $<$ Ring $>$, 77
aerobus::i32, 62	mul_t
aerobus::i64, 69	aerobus, 29
aerobus::polynomial $<$ Ring $>$, 80	aerobus::i32, 61
aerobus::Quotient $<$ Ring, $X >$, 85	aerobus::i64, 68
aerobus::zpz, 111	aerobus::polynomial $<$ Ring $>$, 77
is_field	aerobus::Quotient $<$ Ring, X $>$, 84
aerobus::i32, 62	aerobus:: $zpz $, 109
aerobus::i64, 69	mulfractions_t
aerobus::polynomial $<$ Ring $>$, 80	aerobus, 29
aerobus::zpz, 111	
is_instantiation_of_v	one
aerobus::internal, 40	aerobus::i32, 61
is_zero_t	aerobus::i64, 68
aerobus::i32::val $<$ x $>$, 92	aerobus::polynomial< Ring >, 77
aerobus::i64::val $< x >$, 94	aerobus::Quotient $<$ Ring, X $>$, 84
aerobus::polynomial< Ring >::val< coeffN >, 103	aerobus::zpz $<$ p $>$, 110
aerobus::polynomial< Ring >::val< coeffN, coeffs	
>, 97	physicist
aerobus::zpz $<$ p $>$::val $<$ x $>$, 101	aerobus::known_polynomials, 41
is_zero_v	pi64
aerobus::polynomial< Ring >::val< coeffN >, 105	aerobus, 30
aerobus::polynomial< Ring >::val< coeffN, coeffs	PI_fraction
>, 99	aerobus, 30
aerobus::zpz $<$ p $>$::val $<$ x $>$, 102	pos_t
	aerobus::i32, 61
length	aerobus::i64, 68
aerobus::type_list< Ts >, 89	aerobus::polynomial < Ring >, 77
aerobus::type_list<>, 91	aerobus::Quotient< Ring, X >, 84
Inp1	aerobus::zpz, 110
aerobus, 27	pos_v
lt_t	aerobus::i32, 62
aerobus::i32, 60	aerobus::i64, 70
aerobus::i64, 67	aerobus::polynomial < Ring >, 80
aerobus::polynomial< Ring >, 76	aerobus::Quotient< Ring, X >, 85
aerobus::zpz, 109	aerobus::zpz, 112
lt_v	pow_t
aerobus::i64, 70	aerobus, 30
aerobus::zpz, 111	pq64
	aerobus, 30
make_frac_polynomial_t	probabilist
aerobus, 28	aerobus::known_polynomials, 41
make_index_sequence_reverse	push_back
aerobus::internal, 40	aerobus::type_list< Ts >, 88
make_int_polynomial_t	aerobus::type_list<>, 90
aerobus, 28	push_front
make_q32_t	aerobus::type_list< Ts >, 89
aerobus, 28	aerobus::type_list<>, 90

q32	aerobus::zpz $<$ p $>$::val $<$ x $>$, 101
aerobus, 30	type
q64	aerobus::ContinuedFraction < a0 >, 47
aerobus, 31	aerobus::ContinuedFraction < a0, rest >, 49
	aerobus::Embed $<$ i32, i64 $>$, 51
raw_t	aerobus::Embed< polynomial< Small >,
aerobus::Quotient $<$ Ring, $X >$::val $<$ $V >$, 100	polynomial< Large > >, 52
README.md, 113	aerobus::Embed< q32, q64 >, 53
remove	aerobus::Embed< Quotient< Ring, X >, Ring >,
aerobus::type_list< Ts >, 89	54
ring_type	aerobus::Embed< Ring, FractionField< Ring >>,
aerobus::polynomial< Ring >::val< coeffN >, 104	55
aerobus::polynomial< Ring >::val< coeffN, coeffs	aerobus::Embed $\langle zpz \langle x \rangle$, i32 \rangle , 56
>, 97	aerobus::polynomial< Ring >::horner_reduction_t<
	P >::inner< index, stop >, 71
simplify_t	aerobus::polynomial< Ring >::horner_reduction_t<
aerobus::polynomial $<$ Ring $>$, 79	P >::inner< stop, stop >, 71
sin	aerobus::polynomial< Ring >::val< coeffN
aerobus, 31	>::coeff_at< index, std::enable_if_t<(index<
sinh	$0 \mid \text{index} > 0 \rangle > , 45$
aerobus, 31	aerobus::polynomial< Ring >::val< coeffN
SQRT2_fraction	>::coeff_at< index, std::enable_if_t<(index==0):
aerobus, 31	>, 46
SQRT3_fraction	aerobus::Quotient< Ring, X >::val< V >, 100
aerobus, 31	aerobus::type_list< Ts >::pop_front, 81
src/aerobus.h, 113	type_at_t
src/examples.h, 206	aerobus::internal, 40
stirling_1_signed_t	dorosaotornat, ro
aerobus, 32	V
stirling_1_unsigned_t	aerobus::i32::val $< x >$, 92
aerobus, 32	aerobus::i64::val< x >, 95
stirling_2_t	aerobus::zpz::val< x >, 102
aerobus, 32	vadd_t
strip	aerobus, 34
aerobus::polynomial< Ring >::val< coeffN >, 104	val
aerobus::polynomial< Ring >::val< coeffN, coeffs	aerobus::ContinuedFraction < a0 >, 48
>, 97	aerobus::ContinuedFraction < a0, rest >, 49
sub_t	value
aerobus, 33	aerobus::is_prime< n >, 72
aerobus::i32, 62	value_at_t
aerobus::i64, 68	aerobus::polynomial< Ring >::val< coeffN >, 104
aerobus::polynomial $<$ Ring $>$, 79	aerobus::polynomial< Ring >::val< coeffN, coeffs
aerobus::zpz $<$ p $>$, 110	>, 97
	vmul_t
tail	aerobus, 34
aerobus::type_list< Ts >::pop_front, 81	
aerobus::type_list< Ts >::split< index >, 86	X
tan	aerobus::polynomial $<$ Ring $>$, 79
aerobus, 33	
tanh	zero
aerobus, 33	aerobus::i32, 62
taylor	aerobus::i64, 69
aerobus, 33	aerobus::polynomial< Ring >, 79
to_string	aerobus::Quotient $<$ Ring, $X>$, 85
aerobus::i32::val $< x >$, 92	aerobus::zpz $<$ p $>$, 110
aerobus::i64::val $< x >$, 94	
aerobus::polynomial< Ring >::val< coeffN >, 104	
aerobus::polynomial< Ring >::val< coeffN, coeffs	
>, 98	