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 = (Application) $	
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

Index	217
10.8 examples/compensated_horner.cpp	214
10.7 examples/polynomials_over_finite_field.cpp	
10.6 examples/make_polynomial.cpp	
10.5 examples/modular_arithmetic.cpp	
10.4 examples/continued_fractions.cpp	
10.3 examples/fp16.cu	
10.2 examples/custom_taylor.cpp	
10.1 examples/hermite.cpp	
10 Examples	209
,	
9.5 examples.h	
9.4 src/examples.h File Reference	
9.3 aerobus.h	
9.2 src/aerobus.h File Reference	
9.1 README.md File Reference	
9 File Documentation	113
8.35.3.6 pos_v	112
8.35.3.5 lt_v	
8.35.3.4 is_field	
8.35.3.3 is_euclidean_domain	
8.35.3.2 gt_v	
8.35.3.1 eq_v	
8.35.3 Member Data Documentation	
8.35.2.14 zero	
8.35.2.13 sub_t	
8.35.2.12 pos_t	
8.35.2.11 one	
8.35.2.10 mul_t	
8.35.2.9 mod_t	
8.35.2.8 lt_t	
8.35.2.7 inner_type	
8.35.2.6 inject_constant_t	
8.35.2.5 gt_t	
8.35.2.4 gcd_t	
8.35.2.3 eq_t	
8.35.2.2 div_t	
8.35.2.1 add_t	
8.35.2 Member Typedef Documentation	

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 $1 + x + x^2 + x^3 + \dots$, users may write:

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

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

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

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

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

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

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

1.3 Operations

1.3.1 Field of fractions

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

6 Introduction

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

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

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

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

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

Which also have an evaluation function, as polynomial do.

1.3.2 Quotient

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

For example, if we want R to be \mathbb{Z} represented as aerobus::i64, we can express arithmetic modulo 17 using: using namespace aerobus; using 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 $cuda_fp16.h$ header, so we had to implement our own constexpr static_cast from int16_t to $_half$ to make integers polynomials work with $_half$. See $_thisbug$.

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

One workaround is to add <code>constexpr</code> modifier on line 5039 of file <code>cuda_fp16.h</code>. Once done, `examples\fp16.cu" compiles and generates proper assembly. Please push to make these bug fixed by NVIDIA. < img src=" https://zenodo. \leftarrow org/badge/499577459.svg" alt="DOI"/>

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 .																				 		- 1	113
src/examples.h																				 		2	207

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, I >

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

• struct insert h
· struct is instantiation of

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

    struct laguerre helper

    struct laguerre helper < 0, I >

    struct laguerre_helper< 1, I >

• struct legendre_helper

    struct legendre helper< 0, I >

• struct legendre_helper< 1, I >

    struct Inp1 coeff

    struct Inp1_coeff< T, 0 >

    struct make_taylor_impl

    struct make taylor impl< T, coeff at, std::integer sequence< size t, ls... > >

    struct pop front h

· struct pow

    struct pow< T, p, n, std::enable_if_t< n==0 >>

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

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

· struct pow scalar
· struct remove h

    struct sh coeff

    struct sh_coeff_helper

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

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

· struct sin_coeff

    struct sin coeff helper

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

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

· struct Split
• struct split h

    struct split_h< 0, L1, L2 >

· struct staticcast

    struct stirling_1_helper

    struct stirling_1_helper< T, 0, 0 >

struct stirling_1_helper< T, 0, n, std::enable_if_t<(n > 0)>>
struct stirling_1_helper< T, n, 0, std::enable_if_t<(n > 0)> >
```

- struct stirling_1_helper< T, n, k, std::enable_if_t<(k > 0) &&(n > 0)> > • struct stirling_2_helper struct stirling 2 helper< T, 0, n, std::enable_if_t<(n > 0)> >
- struct stirling 2 helper< T, n, 0, std::enable_if_t<(n > 0)> >
- struct stirling_2_helper< T, n, k, std::enable_if_t<(k > 0) &&(n > 0) &&(k < n)> >
- struct stirling_2_helper< T, n, n, std::enable_if_t<(n >=0)>>
- struct tan_coeff
- · struct tan coeff helper
- struct tan_coeff_helper< T, i, std::enable_if_t<(i % 2) !=0 >>
- struct tan_coeff_helper< T, i, std::enable_if_t<(i % 2)==0 >>
- struct tanh coeff
- struct tanh_coeff_helper
- struct tanh_coeff_helper< T, i, std::enable_if_t<(i % 2) !=0 >>
- struct tanh_coeff_helper< T, i, std::enable_if_t<(i % 2)==0 >>
- · struct touchard coeff
- struct type at
- struct type_at< 0, T, Ts... >
- struct vadd
- struct vadd< v1 >
- struct vadd< v1, vals... >
- struct vmul
- struct vmul< v1 >
- struct vmul< v1, vals... >

Typedefs

- template<size_t i, typename... Ts> using type_at_t = typename type_at< i, Ts... >::type
- template<std::size_t N> $using \ make_index_sequence_reverse = decltype(index_sequence_reverse(std::make_index_sequence < N \\$ >{}))

Functions

• template<std::size_t... ls> constexpr auto index sequence reverse (std::index sequence < ls... > const &) -> decltype(std::index ← sequence < sizeof...(Is) - 1U - Is... >{})

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

8.8.1 Member Function Documentation

8.8.1.1 func()

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

· src/aerobus.h

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

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

Static Public Member Functions

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

8.9.1 Member Function Documentation

8.9.1.1 func()

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

• src/aerobus.h

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

embedding - struct forward declaration

8.10.1 Detailed Description

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

embedding - struct forward declaration

Template Parameters

Small	a ring which can be embedded in Large
Large	a ring in which Small can be embedded
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 float 2 1; // ceil(mantissa/2)
00318
00319
                   static constexpr float shift = (1 \ll r) + 1;
00320
00321
               template<typename T>
00322
00323
               struct Split {
                  static constexpr INLINED DEVICE void func(T a, T *x, T *y) {
00324
                       T z = a * FloatLayout<T>::shift;
*x = z - (z - a);
00325
00326
00327
                       *y = a - *x;
00328
00329
               } ;
00330
               #ifdef WITH_CUDA_FP16
00331
00332
               template<>
00333
               struct Split<__half> {
00334
                   static constexpr INLINED DEVICE void func(__half a, __half *x, _
                                                                                          _half *y) {
00335
                      __half z = a * __half_raw(0x5280); // TODO(JeWaVe): check this value
00336
                        *x = z - (z - a);
                        *y = a - *x;
00337
00338
                   }
00339
               } ;
00340
00341
               template<>
00342
               struct Split<__half2> {
                  static constexpr INLINED DEVICE void func(_half2 a, _half2 *x, _half2 *y) {
    _half2 z = a * _half2(_half_raw(0x5280), _half_raw(0x5280)); // TODO(JeWaVe):
00343
00344
     check this value
00345
                       *x = z - (z - a);
00346
                        *y = a - *x;
              };
00347
00348
               #endif
00349
00350
00351
               template<typename T>
               static constexpr INLINED DEVICE void two_sum(T a, T b, T *x, T *y) {
00352
00353
                  *x = a + b;
                   T z = *x - a;

*y = (a - (*x - z)) + (b - z);
00354
00355
00356
              }
00357
00358
               template<typename T>
00359
               static constexpr INLINED DEVICE void two_prod(T a, T b, T *x, T *y) {
00360
                   *x = a * b;
00361
                   #ifdef __clang_
                   *y = fma_helper<T>::eval(a, b, -*x);
00362
00363
                   #else
00364
                   T ah, al, bh, bl;
00365
                   Split<T>::func(a, &ah, &al);
00366
                   Split<T>::func(b, &bh, &bl);
00367
                   *y = al * bl - (((*x - ah * bh) - al * bh) - ah * bl);
                   #endif
00368
00369
               }
00370
00371
               template<typename T, size_t N>
00372
               static INLINED DEVICE T horner(T *p1, T *p2, T x) {
                   Tr = p1[0] + p2[0];

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

r = r * x + p1[N - i] + p2[N - i];
00373
00374
00375
00376
                   }
00377
00378
                   return r;
00379
            // namespace internal
00380
00381 } // namespace aerobus
```

```
00382
00383 // utilities
00384 namespace aerobus {
00385
          namespace internal {
00386
              template<template<typename...> typename TT, typename T>
00387
               struct is instantiation of : std::false type { };
00388
00389
               template<template<typename...> typename TT, typename... Ts>
00390
               struct is_instantiation_of<TT, TT<Ts...» : std::true_type { };</pre>
00391
               template<template<typename ...> typename TT, typename T>
inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00392
00393
00394
00395
               template <int64_t i, typename T, typename... Ts>
00396
               struct type_at {
                   static_assert(i < sizeof...(Ts) + 1, "index out of range");
using type = typename type_at<i - 1, Ts...>::type;
00397
00398
00399
               };
00400
00401
               template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00402
                   using type = T;
00403
00404
00405
               template <size_t i, typename... Ts>
00406
               using type_at_t = typename type_at<i, Ts...>::type;
00407
00408
00409
               template<size_t n, size_t i, typename E = void>
00410
               struct _is_prime {};
00411
00412
               template<size t i>
00413
               struct _is_prime<0, i> {
00414
                  static constexpr bool value = false;
00415
               };
00416
               template<size_t i>
00417
00418
               struct _is_prime<1, i> {
                   static constexpr bool value = false;
00420
00421
00422
               template<size_t i>
               struct _is_prime<2, i> {
00423
                  static constexpr bool value = true;
00424
00425
00426
00427
               template<size_t i>
00428
               struct _is_prime<3, i> {
                   static constexpr bool value = true;
00429
00430
00431
00432
               template<size_t i>
00433
               struct _is_prime<5, i> {
00434
                   static constexpr bool value = true;
00435
00436
00437
               template<size t i>
               struct _is_prime<7, i> {
00438
00439
                   static constexpr bool value = true;
00440
00441
00442
               template<size_t n, size_t i>
               struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)» {
00443
00444
                   static constexpr bool value = false;
00445
00446
00447
               template<size_t n, size_t i>
               struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)» {
    static constexpr bool value = false;</pre>
00448
00449
00450
00451
00452
               template<size_t n, size_t i>
00453
               struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n) \gg {
00454
                   static constexpr bool value = true;
00455
00456
               template<size_t n, size_t i>
00457
00458
               struct _is_prime<n, i, std::enable_if_t<(
00459
                 n % i == 0 &&
                   n >= 9 &&
00460
                   n % 3 != 0 &&
00461
                   n % 2 != 0 &&
00462
                   i * i > n)» {
00463
00464
                   static constexpr bool value = true;
00465
00466
00467
               template<size_t n, size_t i>
00468
               struct _is_prime<n, i, std::enable_if_t<(</pre>
```

```
n % (i+2) == 0 &&
                   n >= 9 &&
00470
00471
                   n % 3 != 0 &&
                   n % 2 != 0 &&
00472
00473
                   i * i <= n) » {
00474
                   static constexpr bool value = true;
00475
               };
00476
00477
               template<size_t n, size_t i>
               struct _is_prime<n, i, std::enable_if_t<(
    n % (i+2) != 0 &&</pre>
00478
00479
                       n % i != 0 &&
00480
00481
                       n >= 9 &&
00482
                       n % 3 != 0 &&
00483
                       n % 2 != 0 &&
00484
                        (i * i \le n)) \gg \{
                   static constexpr bool value = _is_prime<n, i+6>::value;
00485
00486
00487
          } // namespace internal
00488
00491
          template<size_t n>
00492
          struct is_prime {
00494
              static constexpr bool value = internal::_is_prime<n, 5>::value;
00495
00496
00500
          template<size_t n>
00501
          static constexpr bool is_prime_v = is_prime<n>::value;
00502
00503
          // gcd
00504
          namespace internal {
00505
               template <std::size_t... Is>
00506
               constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
00507
                   -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
00508
00509
               template <std::size_t N>
00510
               using make_index_sequence_reverse
00511
                   = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00518
               template<typename Ring, typename E = void>
00519
               struct gcd;
00520
00521
               template<typename Ring>
               struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
00522
00523
                   template<typename A, typename B, typename E = void>
00524
                   struct gcd_helper {};
00525
00526
                   // B = 0, A > 0
                   template<typename A, typename B>
struct gcd_helper<A, B, std::enable_if_t<</pre>
00527
00528
                       ((B::is_zero_t::value) &&
00529
00530
                           (Ring::template gt_t<A, typename Ring::zero>::value))» {
00531
                       using type = A;
00532
                   } ;
00533
                   // B = 0, A < 0
00534
                   template<typename A, typename B>
00535
00536
                   struct gcd_helper<A, B, std::enable_if_t<
00537
                       ((B::is_zero_t::value) &&
00538
                           !(Ring::template gt_t<A, typename Ring::zero>::value))» {
00539
                       using type = typename Ring::template sub_t<typename Ring::zero, A>;
00540
                   }:
00541
00542
                   // B != 0
00543
                   template<typename A, typename B>
struct gcd_helper<A, B, std::enable_if_t<</pre>
00544
00545
                       (!B::is_zero_t::value)
00546
                       » {
                   private: // NOLINT
00547
00548
                       // A / B
                       using k = typename Ring::template div_t<A, B>;
00549
00550
                            - (A/B) *B = A % B
00551
                       using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
00552
00553
                   public:
00554
                       using type = typename gcd_helper<B, m>::type;
00555
00556
00557
                   template<typename A, typename B>
00558
                   using type = typename gcd_helper<A, B>::type;
00559
              };
          } // namespace internal
00560
00561
00562
          // vadd and vmul
00563
          namespace internal {
00564
               template<typename... vals>
00565
               struct vmul {};
00566
```

```
00567
              template<typename v1, typename... vals>
00568
              struct vmul<v1, vals...> {
00569
                 using type = typename v1::enclosing_type::template mul_t<v1, typename
     vmul<vals...>::type>;
00570
            };
00571
00572
              template<typename v1>
00573
              struct vmul<v1> {
00574
                using type = v1;
00575
             };
00576
00577
             template<typename... vals>
00578
             struct vadd {};
00579
              template<typename v1, typename... vals>
00580
00581
              struct vadd<v1, vals...> {
                 using type = typename v1::enclosing_type::template add_t<v1, typename
00582
     vadd<vals...>::type>;
00583
             };
00584
00585
              template<typename v1>
00586
              struct vadd<v1> {
                using type = v1;
00587
00588
00589
         } // namespace internal
00590
00593
          template<typename T, typename A, typename B>
00594
          using gcd_t = typename internal::gcd<T>::template type<A, B>;
00595
00599
          template<typename... vals>
00600
          using vadd_t = typename internal::vadd<vals...>::type;
00601
00605
          template<typename... vals>
00606
          using vmul_t = typename internal::vmul<vals...>::type;
00607
00611
          template<typename val>
00612
          requires IsEuclideanDomain<typename val::enclosing type>
00613
         using abs_t = std::conditional_t<
00614
                         val::enclosing_type::template pos_v<val>,
                          val, typename val::enclosing_type::template
00615
      sub_t<typename val::enclosing_type::zero, val>>;
00616 } // namespace aerobus
00617
00618 // embedding
00619 namespace aerobus {
00624
        template<typename Small, typename Large, typename E = void>
00625
          struct Embed:
00626 } // namespace aerobus
00627
00628 namespace aerobus {
00633
         template<typename Ring, typename X>
00634
          requires IsRing<Ring>
00635
          struct Quotient {
00638
              template <typename V>
              struct val {
00639
00640
              public:
00641
                 using raw_t = V;
                  using type = abs_t<typename Ring::template mod_t<V, X>>;
00642
00643
00644
00646
             using zero = val<typename Ring::zero>;
00647
00649
              using one = val<typename Ring::one>;
00650
00654
              template<typename v1, typename v2>
00655
              using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00656
00660
              template<typename v1, typename v2>
00661
              using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00662
00666
              template<typename v1, typename v2>
00667
              using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00668
00672
              template<typename v1, typename v2>
00673
              using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00674
00678
              template<typename v1, typename v2>
00679
              using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00680
00684
              template<typename v1, typename v2>
00685
              static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00686
00690
              template<typename v1>
00691
              using pos_t = std::true_type;
00692
00696
              template<typename v>
00697
              static constexpr bool pos v = pos t<v>::value;
```

```
00700
                static constexpr bool is_euclidean_domain = true;
00701
00705
                template<auto x>
00706
                using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00707
00711
                template<typename v>
00712
                using inject_ring_t = val<v>;
00713
00714
           template<typename Ring, typename X>
00718
           struct Embed<Quotient<Ring, X>, Ring> {
00719
00722
                template<typename val>
00723
               using type = typename val::raw_t;
00724
00725 }
          // namespace aerobus
00726
00727 // type_list
00728 namespace aerobus {
00730
           template <typename... Ts>
00731
           struct type_list;
00732
00733
           namespace internal {
00734
               template <typename T, typename... Us>
00735
                struct pop_front_h {
    using tail = type_list<Us...>;
00736
00737
                    using head = T;
00738
00739
00740
                template <size_t index, typename L1, typename L2>
00741
                struct split_h {
00742
                private:
00743
                    static_assert(index <= L2::length, "index ouf of bounds");</pre>
                    using a = typename L2::pop_front::type;
using b = typename L2::pop_front::tail;
00744
00745
00746
                    using c = typename L1::template push_back<a>;
00747
00748
                   using head = typename split_h<index - 1, c, b>::head;
using tail = typename split_h<index - 1, c, b>::tail;
00749
00750
00751
00752
                template <typename L1, typename L2>
struct split_h<0, L1, L2> {
00753
00754
00755
                   using head = L1;
                    using tail = L2;
00756
00757
00758
00759
                template <size_t index, typename L, typename T>
00760
                struct insert h {
00761
                    static_assert(index <= L::length, "index ouf of bounds");</pre>
00762
                    using s = typename L::template split<index>;
00763
                    using left = typename s::head;
                    using right = typename s::tail;
00764
                    using 11 = typename left::template push_back<T>;
using type = typename l1::template concat<right>;
00765
00766
00767
00768
00769
                template <size_t index, typename L>
00770
                struct remove_h {
00771
                    using s = typename L::template split<index>;
                    using left = typename s::head;
using right = typename s::tail;
00772
00773
00774
                    using rr = typename right::pop_front::tail;
00775
                    using type = typename left::template concat<rr>;
00776
00777
           } // namespace internal
00778
00781
           template <typename... Ts>
00782
           struct type_list {
00783
00784
               template <typename T>
00785
               struct concat_h;
00786
00787
                template <typename... Us>
00788
                struct concat_h<type_list<Us...» {
00789
                    using type = type_list<Ts..., Us...>;
00790
00791
00792
            public:
00794
               static constexpr size t length = sizeof...(Ts);
00795
00798
                template <typename T>
00799
                using push_front = type_list<T, Ts...>;
00800
               template <size_t index>
using at = internal::type_at_t<index, Ts...>;
00803
00804
```

```
00807
              struct pop_front {
                  using type = typename internal::pop_front_h<Ts...>::head;
using tail = typename internal::pop_front_h<Ts...>::tail;
00809
00811
00812
00813
00816
              template <typename T>
00817
              using push_back = type_list<Ts..., T>;
00818
00821
              template <typename U>
00822
              using concat = typename concat_h<U>::type;
00823
00826
              template <size_t index>
00827
              struct split {
00828
              private:
00829
                  using inner = internal::split_h<index, type_list<>, type_list<Ts...»;
00830
00831
               public:
00832
                 using head = typename inner::head;
00833
                  using tail = typename inner::tail;
00834
00835
00839
              template <typename T, size_t index>
00840
              using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00841
00844
              template <size_t index>
00845
              using remove = typename internal::remove_h<index, type_list<Ts...»::type;</pre>
00846
         } ;
00847
00849
          template <>
00850
          struct type_list<> {
00851
              static constexpr size_t length = 0;
00852
00853
              template <typename T>
00854
              using push_front = type_list<T>;
00855
00856
              template <typename T>
              using push_back = type_list<T>;
00858
00859
              template <typename U>
00860
              using concat = U;
00861
00862
              // TODO(jewave): assert index == 0
00863
              template <typename T, size_t index>
              using insert = type_list<T>;
00864
00865
00866 } // namespace aerobus
00867
00868 // i16
00869 #ifdef WITH_CUDA_FP16
00870 // i16
00871 namespace aerobus {
00873
         struct i16 {
00874
            using inner_type = int16_t;
00877
              template<int16_t x>
00878
              struct val {
                 using enclosing_type = i16;
00882
                  static constexpr int16_t v = x;
00883
00886
                  template<typename valueType>
                  static constexpr INLINED DEVICE valueType get() {
00887
                       return internal::template int16_convert_helper<valueType, x>::value();
00888
00889
00890
00892
                  using is_zero_t = std::bool_constant<x == 0>;
00893
00895
                  static std::string to_string() {
00896
                      return std::to_string(x);
00897
                  }
00898
              };
00899
00901
              using zero = val<0>;
              using one = val<1>;
00903
00905
              static constexpr bool is_field = false;
              static constexpr bool is_euclidean_domain = true;
00907
00910
              template<auto x>
00911
              using inject_constant_t = val<static_cast<int16_t>(x)>;
00912
00913
              template<typename v>
00914
              using inject_ring_t = v;
00915
00916
           private:
00917
              template<typename v1, typename v2>
00918
              struct add {
00919
                  using type = val<v1::v + v2::v>;
00920
              };
00921
```

```
template<typename v1, typename v2>
00923
00924
                  using type = val<v1::v - v2::v>;
00925
00926
00927
              template<typename v1, typename v2>
00928
              struct mul {
00929
                  using type = val<v1::v* v2::v>;
00930
00931
00932
              template<typename v1, typename v2>
00933
              struct div {
00934
                 using type = val<v1::v / v2::v>;
00935
00936
00937
              template<typename v1, typename v2>
00938
              struct remainder {
                 using type = val<v1::v % v2::v>;
00939
00940
00941
00942
              template<typename v1, typename v2>
00943
              struct qt {
                 using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00944
00945
00946
00947
              template<typename v1, typename v2>
00948
00949
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00950
00951
00952
              template<typename v1, typename v2>
00953
              struct eq {
00954
                 using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00955
00956
00957
              template<typename v1>
00958
              struct pos {
                  using type = std::bool_constant<(v1::v > 0)>;
00960
              };
00961
00962
           public:
00967
              template<typename v1, typename v2> \,
00968
              using add t = typename add<v1, v2>::type;
00969
00974
              template<typename v1, typename v2>
00975
              using sub_t = typename sub<v1, v2>::type;
00976
00981
              template<typename v1, typename v2>
00982
              using mul_t = typename mul<v1, v2>::type;
00983
00988
              template<typename v1, typename v2>
00989
              using div_t = typename div<v1, v2>::type;
00990
00995
              template<typename v1, typename v2>
00996
              using mod_t = typename remainder<v1, v2>::type;
00997
01002
              template<typename v1, typename v2>
01003
              using gt_t = typename gt<v1, v2>::type;
01004
01009
              template<typename v1, typename v2>
01010
              using lt_t = typename lt<v1, v2>::type;
01011
01016
              template<typename v1, typename v2>
01017
              using eq_t = typename eq<v1, v2>::type;
01018
01022
              template<typename v1, typename v2>
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01023
01024
01029
              template<typename v1, typename v2>
01030
              using gcd_t = gcd_t<i16, v1, v2>;
01031
01035
              template < typename v >
01036
              using pos_t = typename pos<v>::type;
01037
01041
              template<typename v>
01042
              static constexpr bool pos_v = pos_t<v>::value;
01043
01044 } // namespace aerobus
01045 #endif
01046
01047 // i32
01048 namespace aerobus {
         struct i32 {
01050
01051
              using inner_type = int32_t;
01054
              template<int32_t x>
01055
              struct val {
01057
                  using enclosing_type = i32;
```

```
static constexpr int32_t v = x;
01060
01063
                  template<typename valueType>
                  static constexpr DEVICE valueType get() {
01064
01065
                      return static_cast<valueType>(x);
01066
01067
01069
                  using is_zero_t = std::bool_constant<x == 0>;
01070
01072
                  static std::string to string() {
01073
                      return std::to_string(x);
01074
01075
              };
01076
01078
              using zero = val<0>;
01080
              using one = val<1>;
              static constexpr bool is_field = false;
01082
01084
              static constexpr bool is_euclidean_domain = true;
01087
              template<auto x>
01088
             using inject_constant_t = val<static_cast<int32_t>(x)>;
01089
01090
              template<typename v>
01091
             using inject_ring_t = v;
01092
01093
          private:
             template<typename v1, typename v2>
01094
01095
              struct add {
01096
                  using type = val<v1::v + v2::v>;
01097
01098
01099
              template<typename v1, typename v2>
01100
              struct sub {
01101
                 using type = val<v1::v - v2::v>;
01102
              } ;
01103
              template<typename v1, typename v2>
01104
01105
              struct mul {
01106
                  using type = val<v1::v* v2::v>;
01107
01108
01109
              template<typename v1, typename v2>
01110
              struct div {
                 using type = val<v1::v / v2::v>;
01111
01112
01113
01114
              template<typename v1, typename v2>
01115
              struct remainder {
01116
                  using type = val<v1::v % v2::v>;
01117
01118
01119
              template<typename v1, typename v2>
01120
01121
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01122
01123
01124
              template<typename v1, typename v2>
01125
              struct lt {
01126
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01127
01128
              template<typename v1, typename v2>
01129
01130
              struct eq {
01131
                 using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01132
01133
01134
              template<typename v1>
01135
              struct pos {
                  using type = std::bool_constant<(v1::v > 0)>;
01136
01137
01138
01139
          public:
01144
              template<typename v1, typename v2>
01145
              using add_t = typename add<v1, v2>::type;
01146
01151
              template<typename v1, typename v2>
01152
              using sub_t = typename sub<v1, v2>::type;
01153
01158
              template<typename v1, typename v2>
01159
              using mul_t = typename mul<v1, v2>::type;
01160
              template<typename v1, typename v2>
01165
01166
              using div_t = typename div<v1, v2>::type;
01167
01172
              template<typename v1, typename v2>
01173
              using mod_t = typename remainder<v1, v2>::type;
01174
01179
              template<tvpename v1, tvpename v2>
```

```
01180
              using gt_t = typename gt<v1, v2>::type;
01181
01186
              template<typename v1, typename v2>
01187
              using lt_t = typename lt<v1, v2>::type;
01188
01193
              template<typename v1, typename v2>
01194
              using eq_t = typename eq<v1, v2>::type;
01195
01199
              template<typename v1, typename v2>
01200
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01201
01206
              template<typename v1, typename v2> \,
              using gcd_t = gcd_t<i32, v1, v2>;
01207
01208
01212
              template<typename v>
01213
              using pos_t = typename pos<v>::type;
01214
01218
              template<typename v>
              static constexpr bool pos_v = pos_t<v>::value;
01220
01221 } // namespace aerobus
01222
01223 // i64
01224 namespace aerobus {
01226
         struct i64 {
            using inner_type = int64_t;
01228
01231
              template<int64_t x>
01232
              struct val {
01234
                 using inner_type = int32_t;
01236
                  using enclosing_type = i64;
01238
                 static constexpr int64_t v = x;
01239
01242
                  template<typename valueType>
01243
                  static constexpr INLINED DEVICE valueType get() {
01244
                      return static_cast<valueType>(x);
01245
01246
01248
                  using is_zero_t = std::bool_constant<x == 0>;
01249
01251
                  static std::string to_string() {
01252
                      return std::to_string(x);
01253
                  }
01254
              }:
01255
01258
              template<auto x>
01259
              using inject_constant_t = val<static_cast<int64_t>(x)>;
01260
01265
              template<typename v>
01266
              using inject_ring_t = v;
01267
01269
              using zero = val<0>;
01271
              using one = val<1>;
01273
              static constexpr bool is_field = false;
01275
              static constexpr bool is_euclidean_domain = true;
01276
01277
          private:
01278
             template<typename v1, typename v2>
01279
              struct add {
01280
                  using type = val<v1::v + v2::v>;
01281
01282
              template<typename v1, typename v2>
01283
01284
              struct sub {
01285
                 using type = val<v1::v - v2::v>;
01286
              } ;
01287
01288
              template<typename v1, typename v2>
01289
              struct mul {
                  using type = val<v1::v* v2::v>;
01290
01291
01292
01293
              template<typename v1, typename v2>
01294
              struct div {
                 using type = val<v1::v / v2::v>;
01295
01296
01297
01298
              template<typename v1, typename v2>
01299
              struct remainder {
01300
                  using type = val<v1::v% v2::v>;
01301
01302
01303
              template<typename v1, typename v2>
01304
01305
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01306
01307
01308
              template<tvpename v1, tvpename v2>
```

```
01309
              struct lt {
01310
                 using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01311
              };
01312
01313
              template<typename v1, typename v2>
01314
              struct ea {
                  using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01315
01316
01317
01318
              template<typename v>
01319
              struct pos {
                 using type = std::bool_constant<(v::v > 0)>;
01320
01321
01322
01323
           public:
01327
              template<typename v1, typename v2>
01328
              using add_t = typename add<v1, v2>::type;
01329
01333
              template<typename v1, typename v2>
01334
              using sub_t = typename sub<v1, v2>::type;
01335
01339
              template<typename v1, typename v2> ^{\circ}
01340
              using mul_t = typename mul<v1, v2>::type;
01341
01346
              template<typename v1, typename v2>
01347
              using div_t = typename div<v1, v2>::type;
01348
01352
              template<typename v1, typename v2>
01353
              using mod_t = typename remainder<v1, v2>::type;
01354
01359
              template<typename v1, typename v2>
01360
              using gt_t = typename gt<v1, v2>::type;
01361
01366
              template<typename v1, typename v2>
01367
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01368
01373
              template<typename v1, typename v2>
01374
              using lt_t = typename lt<v1, v2>::type;
01375
              template<typename v1, typename v2>
static constexpr bool lt_v = lt_t<v1, v2>::value;
01380
01381
01382
              template<typename v1, typename v2>
using eq_t = typename eq<v1, v2>::type;
01387
01388
01389
01394
              template<typename v1, typename v2>
01395
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01396
              template<typename v1, typename v2>
01401
              using gcd_t = gcd_t<i64, v1, v2>;
01402
01403
01407
              template<typename v>
01408
              using pos_t = typename pos<v>::type;
01409
              template<typename v>
01413
              static constexpr bool pos_v = pos_t<v>::value;
01414
01415
         };
01416
01418
          template<>
01419
          struct Embed<i32, i64> {
01422
              template<typename val>
              using type = i64::val<static_cast<int64_t>(val::v)>;
01423
01424
01425 } // namespace aerobus
01426
01427 // z/pz
01428 namespace aerobus {
          template<int32_t p>
01434
01435
          struct zpz {
              using inner_type = int32_t;
01438
01441
              template<int32_t x>
01442
              struct val {
                  using enclosing_type = zpz;
01444
01446
                  static constexpr int32_t v = x % p;
01447
01450
                  template<typename valueType>
                  static constexpr INLINED DEVICE valueType get() {
01451
01452
                       return static_cast<valueType>(x % p);
                  }
01453
01454
01456
                  using is_zero_t = std::bool_constant<v == 0>;
01457
01459
                  static constexpr bool is_zero_v = v == 0;
01460
                  static std::string to string() {
01463
01464
                       return std::to string(x % p);
```

```
01465
                  }
01466
01467
01470
              template<auto x>
              using inject_constant_t = val<static_cast<int32_t>(x)>;
01471
01472
01474
              using zero = val<0>;
01475
01477
              using one = val<1>;
01478
              static constexpr bool is_field = is_prime::value;
01480
01481
01483
             static constexpr bool is_euclidean_domain = true;
01484
01485
          private:
01486
              template<typename v1, typename v2>
01487
              struct add {
01488
                 using type = val<(v1::v + v2::v) % p>;
01489
01490
01491
              template<typename v1, typename v2>
01492
              struct sub {
                 using type = val<(v1::v - v2::v) % p>;
01493
01494
01495
01496
              template<typename v1, typename v2>
01497
01498
                 using type = val<(v1::v* v2::v) % p>;
01499
01500
01501
              template<typename v1, typename v2>
01502
              struct div {
01503
                 using type = val<(v1::v% p) / (v2::v % p)>;
01504
              };
01505
01506
              template<typename v1, typename v2>
01507
              struct remainder {
                  using type = val<(v1::v% v2::v) % p>;
01509
01510
01511
              template<typename v1, typename v2>
01512
              struct qt {
                 using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01513
01514
01515
01516
              template<typename v1, typename v2>
01517
              struct lt {
01518
                  01519
01520
              template<typename v1, typename v2>
01522
              struct eq {
01523
                  using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01524
01525
01526
              template<typename v1>
01527
              struct pos {
01528
                 using type = std::bool_constant<(v1::v > 0)>;
01529
01530
01531
          public:
01535
              template<typename v1, typename v2>
01536
              using add_t = typename add<v1, v2>::type;
01537
01541
              template<typename v1, typename v2>
01542
              using sub_t = typename sub<v1, v2>::type;
01543
01547
              template<typename v1, typename v2>
01548
              using mul t = typename mul<v1, v2>::type;
01553
              template<typename v1, typename v2>
01554
              using div_t = typename div<v1, v2>::type;
01555
01559
              template<typename v1, typename v2>
              using mod_t = typename remainder<v1, v2>::type;
01560
01561
01565
              template<typename v1, typename v2>
01566
              using gt_t = typename gt<v1, v2>::type;
01567
              template<typename v1, typename v2> static constexpr bool gt_v = gt_t<v1, v2>::value;
01571
01572
01573
01577
              template<typename v1, typename v2>
01578
              using lt_t = typename lt<v1, v2>::type;
01579
              template<typename v1, typename v2>
static constexpr bool lt_v = lt_t<v1, v2>::value;
01583
01584
```

```
01589
              template<typename v1, typename v2>
01590
              using eq_t = typename eq<v1, v2>::type;
01591
01595
              template<typename v1, typename v2> \,
01596
              static constexpr bool eq_v = eq_t<v1, v2>::value;
01597
01601
              template<typename v1, typename v2>
01602
              using gcd_t = gcd_t < i32, v1, v2>;
01603
01606
              template<typename v1>
01607
              using pos_t = typename pos<v1>::type;
01608
01611
              template<typename v>
01612
              static constexpr bool pos_v = pos_t<v>::value;
01613
         };
01614
          template<int32 t x>
01617
01618
          struct Embed<zpz<x>, i32> {
              template <typename val>
01621
              using type = i32::val<val::v>;
01622
01623
01624 } // namespace aerobus
01625
01626 // polynomial
01627 namespace aerobus {
01628
          // coeffN x^N + ...
01633
          template<typename Ring>
01634
          requires IsEuclideanDomain<Ring>
01635
          struct polynomial {
01636
              static constexpr bool is_field = false;
01637
              static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01638
              template<typename P>
01641
01642
              struct horner_reduction_t {
                  template<size_t index, size_t stop>
01643
                  struct inner {
01644
01645
                      template<typename accum, typename x>
01646
                      using type = typename horner_reduction_t<P>::template inner<index + 1, stop>
01647
                          ::template type<
01648
                               typename Ring::template add_t<</pre>
                                   typename Ring::template mul t<x, accum>,
01649
01650
                                   typename P::template coeff_at_t<P::degree - index>
01651
                              >, x>;
01652
                  };
01653
01654
                  template<size_t stop>
01655
                  struct inner<stop, stop> {
                      template<typename accum, typename x> using type = accum;
01656
01657
01658
                  };
01659
              };
01660
01664
              template<typename coeffN, typename... coeffs>
01665
              struct val {
01667
                  using ring type = Ring;
                  using enclosing_type = polynomial<Ring>;
01669
01671
                  static constexpr size_t degree = sizeof...(coeffs);
01673
                  using aN = coeffN;
01675
                  using strip = val<coeffs...>;
                  using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01677
01679
                  static constexpr bool is_zero_v = is_zero_t::value;
01680
01681
01682
                  template<size_t index, typename E = void>
01683
                  struct coeff_at {};
01684
01685
                  template<size t index>
01686
                  struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))» {</pre>
                      using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01688
01689
01690
                  template<size_t index>
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))» {
01691
01692
                      using type = typename Ring::zero;
01693
01694
01695
               public:
01698
                  template<size_t index>
                  using coeff_at_t = typename coeff_at<index>::type;
01699
01700
01703
                  static std::string to_string() {
01704
                      return string_helper<coeffN, coeffs...>::func();
01705
01706
01711
                  template<typename arithmeticType>
01712
                  static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
```

```
#ifdef WITH CUDA FP16
                      arithmeticType start;
01714
01715
                      if constexpr (std::is_same_v<arithmeticType, __half2>) {
01716
                          start = \underline{\quad}half2(0, 0);
01717
                      } else {
01718
                          start = static cast<arithmeticType>(0);
01719
01720
01721
                      arithmeticType start = static_cast<arithmeticType>(0);
01722
                      #endif
01723
                      return horner_evaluation<arithmeticType, val>
                              ::template inner<0, degree + 1>
01724
01725
                              ::func(start, x);
01726
                  }
01727
01740
                  template<typename arithmeticType>
                  static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01741
01742
                      return compensated_horner<arithmeticType, val>::func(x);
01744
01745
                  template<typename x>
01746
                  using value_at_t = horner_reduction_t<val>
01747
                      ::template inner<0, degree + 1>
01748
                      ::template type<typename Ring::zero, x>;
01749
              };
01750
              template<typename coeffN>
01753
01754
              struct val<coeffN> {
                  using ring_type = Ring;
01756
01758
                  using enclosing_type = polynomial<Ring>;
01760
                  static constexpr size_t degree = 0;
01761
                  using aN = coeffN;
01762
                  using strip = val<coeffN>;
01763
                  using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01764
                  static constexpr bool is_zero_v = is_zero_t::value;
01765
01766
01767
                  template<size_t index, typename E = void>
01768
                  struct coeff_at {};
01769
01770
                  template<size_t index>
                  struct coeff_at<index, std::enable_if_t<(index == 0)» {</pre>
01771
01772
                      using type = aN;
01773
01774
01775
                  template<size_t index>
01776
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01777
                      using type = typename Ring::zero;
01778
01779
01780
                  template<size_t index>
01781
                  using coeff_at_t = typename coeff_at<index>::type;
01782
01783
                  static std::string to_string() {
01784
                      return string_helper<coeffN>::func();
01785
01786
01787
                  template<typename arithmeticType>
01788
                  static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01789
                      return coeffN::template get<arithmeticType>();
01790
01791
01792
                  template<typename arithmeticType>
01793
                  static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01794
                      return coeffN::template get<arithmeticType>();
01795
01796
01797
                  template<typename x>
01798
                  using value at t = coeffN:
01799
              };
01800
01802
              using zero = val<typename Ring::zero>;
              using one = val<typename Ring::one>;
01804
01806
              using X = val<typename Ring::one, typename Ring::zero>;
01807
01808
01809
              template<typename P, typename E = void>
01810
              struct simplify;
01811
01812
              template <typename P1, typename P2, typename I>
01813
              struct add low;
01814
01815
              template<typename P1, typename P2>
01816
              struct add {
01817
                  using type = typename simplify<typename add_low<
                  P1,
01818
01819
                  P2.
```

```
internal::make_index_sequence_reverse<
                   std::max(P1::degree, P2::degree) + 1
01821
01822
                   »::type>::type;
01823
              };
01824
01825
              template <typename P1, typename P2, typename I>
01826
              struct sub_low;
01827
01828
              template <typename P1, typename P2, typename I>
01829
              struct mul_low;
01830
01831
              template<typename v1, typename v2>
01832
              struct mul {
01833
                       using type = typename mul_low<
01834
01835
                           v2,
01836
                           internal::make_index_sequence_reverse<
01837
                           v1::degree + v2::degree + 1
01838
                           »::type;
01839
              };
01840
01841
              template<typename coeff, size_t deg>
01842
              struct monomial;
01843
01844
              template<typename v, typename E = void>
              struct derive_helper {};
01845
01846
01847
              template<typename v>
01848
              struct derive_helper<v, std::enable_if_t<v::degree == 0» {</pre>
01849
                  using type = zero;
01850
01851
01852
              template<typename v>
01853
              struct derive_helper<v, std::enable_if_t<v::degree != 0» {
01854
                  using type = typename add<
01855
                       typename derive_helper<typename simplify<typename v::strip>::type>::type,
01856
                       typename monomial<
                           typename Ring::template mul_t<
01858
                               typename v::aN,
01859
                               typename Ring::template inject_constant_t<(v::degree)>
01860
                           v::degree - 1
01861
                       >::type
01862
01863
                  >::type;
01864
              };
01865
01866
              template<typename v1, typename v2, typename E = void>
01867
              struct eq_helper {};
01868
01869
              template<typename v1, typename v2>  
              struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
01871
                  using type = std::false_type;
01872
01873
01874
01875
              template<typename v1, typename v2>
01876
              struct eq_helper<v1, v2, std::enable_if_t<
01877
                  v1::degree == v2::degree &&
01878
                   (v1::degree != 0 || v2::degree != 0) &&
01879
                   std::is_same<
                  typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01880
                  std::false_type
01881
01882
                  >::value
01883
01884
              > {
01885
                  using type = std::false_type;
01886
              };
01887
              template<typename v1, typename v2>
struct eq_helper<v1, v2, std::enable_if_t<
    v1::degree == v2::degree &&</pre>
01888
01890
                   (v1::degree != 0 || v2::degree != 0) &&
01891
01892
                  std::is_same<
01893
                  typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01894
                  std::true_type
01895
                  >::value
01896
              » {
01897
                   using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01898
              } ;
01899
01900
              template<typename v1, typename v2>
              v1::degree == v2::degree &&
01901
01902
01903
                   (v1::degree == 0)
01904
01905
                  using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01906
              };
```

```
01908
              template<typename v1, typename v2, typename E = void>
01909
              struct lt_helper {};
01910
01911
              template<typename v1, typename v2>
struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {</pre>
01912
01913
                  using type = std::true_type;
01914
01915
01916
              template<typename v1, typename v2>
              struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {</pre>
01917
                 using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01918
01919
01920
01921
              template<typename v1, typename v2>
01922
              01923
                  using type = std::false_type;
01924
01925
01926
              template<typename v1, typename v2, typename E = void>
01927
              struct gt_helper {};
01928
              01929
01930
01931
                  using type = std::true_type;
01932
01933
              template<typename v1, typename v2>
struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
    using type = std::false_type;</pre>
01934
01935
01936
01937
01938
01939
              template<typename v1, typename v2>
01940
              struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {</pre>
01941
                  using type = std::false_type;
01942
01943
01944
              // when high power is zero : strip
01945
              template<typename P>
01946
              struct simplify<P, std::enable_if_t<
01947
                  std::is_same<
01948
                  typename Ring::zero,
                  typename P::aN
01949
01950
                  >::value && (P::degree > 0)
01951
              » {
01952
                  using type = typename simplify<typename P::strip>::type;
01953
01954
              // otherwise : do nothing
01955
01956
              template<tvpename P>
01957
              struct simplify<P, std::enable_if_t<
                 !std::is_same<
01958
01959
                  typename Ring::zero,
01960
                  typename P::aN
                  >::value && (P::degree > 0)
01961
01962
              » {
01963
                  using type = P;
01964
              } ;
01965
              \ensuremath{//} do not simplify constants
01966
01967
              template<typename P>
01968
              struct simplify<P, std::enable_if_t<P::degree == 0» {</pre>
01969
                  using type = P;
01970
01971
01972
              // addition at
01973
              template<typename P1, typename P2, size_t index>
01974
              struct add at {
01975
                  using type =
01976
                      typename Ring::template add_t<
01977
                         typename P1::template coeff_at_t<index>,
01978
                          typename P2::template coeff_at_t<index»;</pre>
01979
              };
01980
              template<typename P1, typename P2, size_t index>
01981
01982
              using add_at_t = typename add_at<P1, P2, index>::type;
01983
01984
              template<typename P1, typename P2, std::size_t... I>
01985
              struct add_low<P1, P2, std::index_sequence<I...» {</pre>
                  using type = val<add_at_t<P1, P2, I>...>;
01986
01987
01988
01989
              // substraction at
01990
              template<typename P1, typename P2, size_t index>
01991
              struct sub_at {
01992
                  using type =
01993
                      typename Ring::template sub t<
```

```
typename P1::template coeff_at_t<index>,
                           typename P2::template coeff_at_t<index»;</pre>
01995
01996
              };
01997
              template<typename P1, typename P2, size_t index>
01998
01999
              using sub_at_t = typename sub_at<P1, P2, index>::type;
02000
02001
              template<typename P1, typename P2, std::size_t... I>
02002
              struct sub_low<P1, P2, std::index_sequence<I...» {</pre>
02003
                  using type = val<sub_at_t<P1, P2, I>...>;
02004
              };
02005
02006
              template<typename P1, typename P2>
02007
              struct sub {
02008
                  using type = typename simplify<typename sub_low<
02009
                  P1,
02010
                  P2.
                  internal::make_index_sequence_reverse<
std::max(P1::degree, P2::degree) + 1</pre>
02011
02012
02013
                  »::type>::type;
02014
02015
              \ensuremath{//} multiplication at
02016
02017
              template<typename v1, typename v2, size_t k, size_t index, size_t stop>
02018
              struct mul_at_loop_helper {
                  using type = typename Ring::template add_t<
02019
02020
                       typename Ring::template mul_t
02021
                      typename v1::template coeff_at_t<index>,
02022
                      typename v2::template coeff_at_t<k - index>
02023
02024
                      typename mul at loop helper<v1, v2, k, index + 1, stop>::type
02025
02026
02027
              template<typename v1, typename v2, size_t k, size_t stop> struct mul_at_loop_helper<v1, v2, k, stop, stop> {
02028
02029
                  using type = typename Ring::template mul_t<
02030
                      typename v1::template coeff_at_t<stop>,
02031
02032
                      typename v2::template coeff_at_t<0»;
02033
              };
02034
02035
              template <typename v1, typename v2, size_t k, typename E = void>
02036
              struct mul at {};
02037
              02038
02039
02040
                  using type = typename Ring::zero;
02041
              };
02042
02043
              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)» {
02044
02045
                  using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
02046
02047
02048
              template<typename P1, typename P2, size_t index>
02049
              using mul_at_t = typename mul_at<P1, P2, index>::type;
02050
02051
              template<typename P1, typename P2, std::size_t... I>
02052
              struct mul_low<P1, P2, std::index_sequence<I...» {
02053
                  using type = val<mul_at_t<P1, P2, I>...>;
02054
02055
02056
              // division helper
02057
              template< typename A, typename B, typename Q, typename R, typename E = void>
02058
              struct div_helper {};
02059
02060
              template<typename A, typename B, typename Q, typename R>
02061
              struct div_helper<A, B, Q, R, std::enable_if_t<
02062
                   (R::degree < B::degree) ||
02063
                   (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
02064
                  using q_type = Q;
02065
                  using mod_type = R;
                  using gcd_type = B;
02066
02067
02068
02069
              template<typename A, typename B, typename Q, typename R>
02070
              struct div_helper<A, B, Q, R, std::enable_if_t<
02071
                   (R::degree >= B::degree) &&
02072
                  !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
               private: // NOLINT
   using rN = typename R::aN;
02073
02074
02075
                  using bN = typename B::aN;
                  using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
     B::degree>::type;
02077
                  using rr = typename sub<R, typename mul<pT, B>::type>::type;
                  using qq = typename add<Q, pT>::type;
02078
02079
```

```
02080
                public:
02081
                  using q_type = typename div_helper<A, B, qq, rr>::q_type;
02082
                   using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
                   using gcd_type = rr;
02083
02084
02085
02086
               template<typename A, typename B>
02087
               struct div {
02088
                static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
                   using q_type = typename div_helper<A, B, zero, A>::q_type; using m_type = typename div_helper<A, B, zero, A>::mod_type;
02089
02090
02091
              };
02092
02093
               template<typename P>
02094
               struct make_unit {
02095
                  using type = typename div<P, val<typename P::aN»::q_type;
02096
               };
02097
02098
               template<typename coeff, size_t deg>
02099
               struct monomial {
                   using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
02100
02101
02102
               template<typename coeff>
02103
02104
               struct monomial<coeff, 0> {
02105
                 using type = val<coeff>;
02106
02107
02108
               template<typename arithmeticType, typename P>
02109
               \verb|struct horner_evaluation| \{
02110
                   template<size_t index, size_t stop>
02111
                   struct inner {
02112
                       static constexpr DEVICE INLINED arithmeticType func(
02113
                            const arithmeticType& accum, const arithmeticType& x) {
02114
                            return horner_evaluation<arithmeticType, P>::template inner<index + 1,</pre>
      stop>::func(
02115
                                internal::fma helper<arithmeticType>::eval(
02116
02117
                                    accum,
                                    P::template coeff_at_t<P::degree - index>::template
02118
      get<arithmeticType>()), x);
02119
02120
                   }:
02121
02122
                   template<size_t stop>
02123
                   struct inner<stop, stop> {
02124
                      static constexpr DEVICE INLINED arithmeticType func(
02125
                           const arithmeticType& accum, const arithmeticType& x) {
02126
                            return accum;
02127
                       }
02128
                   };
02129
              };
02130
02131
               template<typename arithmeticType, typename P>
               struct compensated_horner {
02132
02133
                  template<int64 t index, int ghost>
                   struct EFTHorner {
02134
02135
                       static INLINED DEVICE void func(
                                arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType
02136
*r) {
                            arithmeticType p;
                            internal::two_prod(*r, x, &p, pi + P::degree - index - 1);
constexpr arithmeticType coeff = P::template coeff_at_t<index>::template
02138
02139
      get<arithmeticType>();
02140
                            internal::two_sum<arithmeticType>(
02141
                                p, coeff,
                                r, sigma + P::degree - index - 1);
02142
                            EFTHorner<index - 1, ghost>::func(x, pi, sigma, r);
02143
02144
                        }
02145
                   };
02146
02147
                   template<int ghost>
                   struct EFTHorner<-1, ghost> {
    static INLINED DEVICE void func(
02148
02149
02150
                                arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType
      *r) {
02151
02152
                   };
02153
                   static INLINED DEVICE arithmeticType func(arithmeticType x) {
02154
02155
                       arithmeticType pi[P::degree], sigma[P::degree];
                        arithmeticType r = P::template coeff_at_t<P::degree>::template get<arithmeticType>();
02156
02157
                       EFTHorner<P::degree - 1, 0>::func(x, pi, sigma, &r);
02158
                       arithmeticType c = internal::horner<arithmeticType, P::degree - 1>(pi, sigma, x);
02159
                        return r + c;
02160
                   }
02161
               };
```

```
02162
              template<typename coeff, typename... coeffs>
02163
02164
              struct string_helper {
02165
                  static std::string func() {
02166
                      std::string tail = string_helper<coeffs...>::func();
std::string result = "";
02167
02168
                       if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
02169
                           return tail;
02170
                       } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
                           if (sizeof...(coeffs) == 1) {
    result += "x";
02171
02172
                           } else {
02173
                               result += "x^" + std::to_string(sizeof...(coeffs));
02174
02175
02176
                       } else {
02177
                          if (sizeof...(coeffs) == 1) {
                               result += coeff::to_string() + " x";
02178
02179
                           } else {
02180
                               result += coeff::to_string()
02181
                                       + " x^" + std::to_string(sizeof...(coeffs));
02182
02183
02184
                       if (!tail.empty()) {
02185
                           if (tail.at(0) != '-') {
    result += " + " + tail;
02186
02187
02188
                           } else {
                               result += " - " + tail.substr(1);
02189
02190
02191
                       }
02192
02193
                       return result;
02194
                  }
02195
              } ;
02196
              template<tvpename coeff>
02197
02198
              struct string helper<coeff> {
02199
                  static std::string func()
02200
                      if (!std::is_same<coeff, typename Ring::zero>::value) {
02201
                           return coeff::to_string();
02202
                      } else {
                           return "";
02203
                       }
02204
02205
                  }
02206
              };
02207
02208
           public:
02211
              template<typename P>
02212
              using simplify_t = typename simplify<P>::type;
02213
02217
              template<typename v1, typename v2>
02218
              using add_t = typename add<v1, v2>::type;
02219
02223
              template<typename v1, typename v2>
02224
              using sub_t = typename sub<v1, v2>::type;
02225
02229
              template<typename v1, typename v2>
02230
              using mul_t = typename mul<v1, v2>::type;
02231
02235
              template<typename v1, typename v2>
02236
              using eq_t = typename eq_helper<v1, v2>::type;
02237
02241
              template<typename v1, typename v2>
02242
              using lt_t = typename lt_helper<v1, v2>::type;
02243
02247
              template<typename v1, typename v2>
02248
              using gt_t = typename gt_helper<v1, v2>::type;
02249
              template<typename v1, typename v2>
02253
02254
              using div_t = typename div<v1, v2>::q_type;
02255
02259
              template<typename v1, typename v2>
02260
              using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
02261
02265
              template<typename coeff, size t deg>
02266
              using monomial_t = typename monomial<coeff, deg>::type;
02267
02270
              template<typename v>
02271
              using derive_t = typename derive_helper<v>::type;
02272
02275
              template<typename v>
              using pos_t = typename Ring::template pos_t<typename v::aN>;
02277
02280
              template<typename v>
02281
              static constexpr bool pos_v = pos_t<v>::value;
02282
02286
              template<tvpename v1, tvpename v2>
```

```
using gcd_t = std::conditional_t<</pre>
02288
                  Ring::is_euclidean_domain,
02289
                   typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
02290
                  void>:
02291
02294
              template<auto x>
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
02295
02296
02299
              template<typename v>
02300
              using inject_ring_t = val<v>;
02301
          };
02302 } // namespace aerobus
02303
02304 // fraction field
02305 namespace aerobus {
02306
          namespace internal {
              template<typename Ring, typename E = void>
02307
02308
              requires IsEuclideanDomain<Ring>
              struct _FractionField {};
02309
02310
02311
              template<typename Ring>
02312
              requires IsEuclideanDomain<Ring>
02313
              struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain» {</pre>
                  static constexpr bool is_field = true;
static constexpr bool is_euclidean_domain = true;
02315
02316
02317
02318
02319
                  template<typename val1, typename val2, typename E = void>
02320
                  struct to_string_helper {};
02321
02322
                  template<typename val1, typename val2>
02323
                  struct to_string_helper <val1, val2,
02324
                       std::enable_if_t<
02325
                       Ring::template eq_t<
02326
                       val2, typename Ring::one
02327
                       >::value
02328
02329
02330
                      static std::string func()
02331
                          return vall::to_string();
02332
02333
                  };
02334
02335
                  template<typename val1, typename val2>
02336
                  struct to_string_helper<val1, val2,
02337
                       std::enable_if_t<
02338
                       !Ring::template eq_t<
02339
                       val2.
                       typename Ring::one
02340
02341
                       >::value
02342
02343
02344
                       static std::string func() {
                          return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
02345
02346
02347
                  };
02348
02349
               public:
02353
                  template<typename val1, typename val2>
                  struct val {
02354
                      using x = val1;
02356
                      using y = val2;
02358
02360
                       using is_zero_t = typename vall::is_zero_t;
02362
                       static constexpr bool is_zero_v = val1::is_zero_t::value;
02363
                       using ring_type = Ring;
02365
                       using enclosing_type = _FractionField<Ring>;
02366
02367
02370
                       static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
02371
02372
                       template<typename valueType, int ghost = 0>
02373
                       struct get_helper {
02374
                           static constexpr INLINED DEVICE valueType get() {
                               return internal::staticcast<valueType, typename</pre>
02375
      ring type::inner type>::template func<x::v>() /
02376
                                   internal::staticcast<valueType, typename ring_type::inner_type>::template
      func<y::v>();
02377
02378
                       };
02379
                       #ifdef WITH_CUDA_FP16
02380
02381
                       template<int ghost>
02382
                       struct get_helper<__half, ghost> {
02383
                           static constexpr INLINED DEVICE
                                                              _half get() {
02384
                              return internal::my_float2half_rn(
02385
                                   internal::staticcast<float, typename ring_type::inner_type>::template
      func<x::v>() /
```

```
02386
                                   internal::staticcast<float, typename ring_type::inner_type>::template
      func<y::v>());
02387
                          }
02388
                      };
02389
                      template<int ghost>
02390
02391
                      struct get_helper<__half2, ghost> {
02392
                          static constexpr INLINED DEVICE __half2 get() {
02393
                             constexpr __half tmp = internal::my_float2half_rn(
02394
                                   internal::staticcast<float, typename ring_type::inner_type>::template
      func<x::v>() /
02395
                                  internal::staticcast<float, typename ring_type::inner_type>::template
      func<v::v>());
02396
                              return __half2(tmp, tmp);
02397
02398
                      };
                       #endif
02399
02400
02404
                      template<typename valueType>
                      static constexpr INLINED DEVICE valueType get() {
02405
02406
                          return get_helper<valueType, 0>::get();
02407
02408
                      static std::string to_string() {
02411
02412
                          return to_string_helper<val1, val2>::func();
02413
02414
02419
                      template<typename arithmeticType>
                      \verb|static constexpr DEVICE INLINED| arithmeticType eval(const arithmeticType \& v) | \{ \\
02420
02421
                          return x::eval(v) / y::eval(v);
02422
02423
                  };
02424
02426
                  using zero = val<typename Ring::zero, typename Ring::one>;
02428
                  using one = val<typename Ring::one, typename Ring::one>;
02429
02432
                  template<typename v>
02433
                  using inject_t = val<v, typename Ring::one>;
02434
02437
                  template<auto x>
02438
                  using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
     Ring::one>;
02439
02442
                  template<typename v>
                  using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
02443
02444
02446
                  using ring_type = Ring;
02447
02448
               private:
02449
                  template<typename v, typename E = void>
02450
                  struct simplify {};
02451
02452
02453
                  template<typename v>
                  struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {
02454
                      using type = typename _FractionField<Ring>::zero;
02455
02456
02457
02458
                  // x != 0
02459
                  template<typename v>
02460
                  struct simplify<v, std::enable if t<!v::x::is zero t::value» {
02461
                   private:
02462
                      using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
                      using newx = typename Ring::template div_t<typename v::x, _gcd>;
02463
02464
                      using newy = typename Ring::template div_t<typename v::y, _gcd>;
02465
02466
                      using posx = std::conditional_t<
                                           !Ring::template pos_v<newy>,
02467
02468
                                           typename Ring::template sub_t<typename Ring::zero, newx>,
02469
                                           newx>;
02470
                      using posy = std::conditional_t<
02471
                                           !Ring::template pos_v<newy>,
02472
                                           typename Ring::template sub_t<typename Ring::zero, newy>,
02473
                                           newv>;
02474
                   public:
02475
                      using type = typename _FractionField<Ring>::template val<posx, posy>;
02476
02477
               public:
02478
02481
                  template<tvpename v>
02482
                  using simplify_t = typename simplify<v>::type;
02483
02484
02485
                  template<typename v1, typename v2>
02486
                  struct add {
02487
                   private:
02488
                      using a = typename Ring::template mul t<typename v1::x, typename v2::v>;
```

```
using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
                      using dividend = typename Ring::template add_t<a, b>;
02490
02491
                      using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02492
                      using g = typename Ring::template gcd_t<dividend, diviser>;
02493
02494
                  public:
02495
                     using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
     diviser»;
02496
02497
02498
                  template<typename v>
02499
                  struct pos {
02500
                      using type = std::conditional t<
02501
                          (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
02502
                          (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
02503
                          std::true_type,
02504
                         std::false_type>;
02505
                 };
02506
02507
                  template<typename v1, typename v2>
02508
                  struct sub {
                  private:
02509
02510
                     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>;
02511
02512
                     using dividend = typename Ring::template sub_t<a, b>;
                     using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02513
02514
                     using g = typename Ring::template gcd_t<dividend, diviser>;
02515
                  public:
02516
02517
                      using type = typename _FractionField<Ring>::template simplify t<val<dividend,
     diviser»:
02518
02519
02520
                  template<typename v1, typename v2>
                  struct mul {
02521
                  private:
02522
                     using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
02523
                     using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02525
02526
                  public:
02527
                     using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
02528
                  }:
02529
02530
                  template<typename v1, typename v2, typename E = void>
02531
                  struct div {};
02532
02533
                  template<typename v1, typename v2>
02534
                  struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
     _FractionField<Ring>::zero>::value»
02535
                  private:
02536
                     using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02537
                     using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02538
                  public:
02539
02540
                     using type = typename _FractionField<Ring>::template simplify_t<val<a, b>;
02541
                  };
02542
02543
                  template<typename v1, typename v2>
02544
                  struct div<v1, v2, std::enable_if_t<
02545
                      std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
02546
                      using type = one;
02547
                  };
02548
02549
                  template<typename v1, typename v2>
02550
                  struct eq
02551
                     using type = std::conditional_t<
02552
                              02553
                              std::is_same<typename simplify_t<vl>::y, typename simplify_t<v2>::y>::value,
02554
                         std::true type,
02555
                         std::false_type>;
02556
                  };
02557
02558
                  template<typename v1, typename v2, typename E = void>
02559
                  struct at:
02560
02561
                  template<typename v1, typename v2>
02562
                  struct gt<v1, v2, std::enable_if_t<
02563
                      (eq<v1, v2>::type::value)
02564
02565
                      using type = std::false type;
02566
                  };
02567
02568
                  template<typename v1, typename v2>
02569
                  struct gt<v1, v2, std::enable_if_t<
02570
                      (!eq<v1, v2>::type::value) &&
02571
                      (!pos<v1>::type::value) && (!pos<v2>::type::value)
02572
```

```
using type = typename gt<
02574
                           typename sub<zero, v1>::type, typename sub<zero, v2>::type
                       >::type;
02575
02576
                  };
02577
02578
                   template<typename v1, typename v2>
                   struct gt<v1, v2, std::enable_if_t<
02579
02580
                       (!eq<v1, v2>::type::value) &&
02581
                       (pos<v1>::type::value) && (!pos<v2>::type::value)
02582
02583
                       using type = std::true_type;
02584
                  };
02585
02586
                   template<typename v1, typename v2>
02587
                   struct gt<v1, v2, std::enable_if_t<
02588
                       (!eq<v1, v2>::type::value) &&
                       (!pos<v1>::type::value) && (pos<v2>::type::value)
02589
02590
                       >> {
02591
                       using type = std::false_type;
02592
                   } ;
02593
02594
                   template<typename v1, typename v2>
                   struct gt<v1, v2, std::enable_if_t<
     (!eq<v1, v2>::type::value) &&
02595
02596
02597
                       (pos<v1>::type::value) && (pos<v2>::type::value)
02598
02599
                       using type = typename Ring::template gt_t<
02600
                           typename Ring::template mul_t<v1::x, v2::y>,
02601
                           typename Ring::template mul_t<v2::y, v2::x>
02602
02603
                   };
02604
02605
                public:
02609
                  template<typename v1, typename v2>
02610
                  using add_t = typename add<v1, v2>::type;
02611
02616
                  template<typename v1, typename v2>
                  using mod_t = zero;
02617
02618
02623
                  template<typename v1, typename v2>
02624
                  using gcd_t = v1;
02625
02629
                  template<typename v1, typename v2>
02630
                  using sub_t = typename sub<v1, v2>::type;
02631
02635
                   template<typename v1, typename v2>
02636
                  using mul_t = typename mul<v1, v2>::type;
02637
02641
                   template<typename v1, typename v2>
02642
                  using div t = typename div<v1, v2>::type;
02643
02647
                   template<typename v1, typename v2>
02648
                   using eq_t = typename eq<v1, v2>::type;
02649
                  template<typename v1, typename v2>
static constexpr bool eq_v = eq<v1, v2>::type::value;
02653
02654
02655
02659
                   template<typename v1, typename v2>
02660
                   using gt_t = typename gt<v1, v2>::type;
02661
02665
                  template<typename v1, typename v2>
02666
                  static constexpr bool gt_v = gt<v1, v2>::type::value;
02667
02670
                   template<typename v1>
02671
                   using pos_t = typename pos<v1>::type;
02672
02675
                  template<typename v>
02676
                   static constexpr bool pos_v = pos_t < v > :: value;
02677
              };
02679
               template<typename Ring, typename E = void>
02680
               requires IsEuclideanDomain<Ring>
02681
              struct FractionFieldImpl {};
02682
               // fraction field of a field is the field itself
02683
02684
               template<typename Field>
02685
               requires IsEuclideanDomain<Field>
02686
               struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {</pre>
02687
                  using type = Field;
                   template<typename v>
02688
02689
                   using inject_t = v;
02690
              } ;
02691
02692
               // fraction field of a ring is the actual fraction field
02693
               template<typename Ring>
               requires IsEuclideanDomain<Ring>
02694
              struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {</pre>
02695
```

```
using type = _FractionField<Ring>;
02697
02698
          } // namespace internal
02699
02702
          template<typename Ring>
02703
          requires IsEuclideanDomain<Ring>
02704
          using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02705
02708
          template<typename Ring>
02709
          struct Embed<Ring, FractionField<Ring» {
02712
               template<typename v>
02713
               using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02714
02715 } // namespace aerobus
02716
02717
02718 // short names for common types
02719 namespace aerobus {
          template<typename X, typename Y>
02724
          requires IsRing<typename X::enclosing_type> &&
02725
               (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02726
          using add_t = typename X::enclosing_type::template add_t<X, Y>;
02727
02731
          template<typename X, typename Y>
requires IsRing<typename X::enclosing_type> &&
02732
02733
              (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02734
          using sub_t = typename X::enclosing_type::template sub_t<X, Y>;
02735
02739
          template<typename X, typename Y>
02740
          requires IsRing<typename X::enclosing_type> &&
               (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02741
02742
          using mul_t = typename X::enclosing_type::template mul_t<X, Y>;
02743
02747
           template<typename X, typename Y>
          requires IsEuclideanDomain<typename X::enclosing_type> &&
   (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02748
02749
02750
          using div_t = typename X::enclosing_type::template div_t<X, Y>;
02751
02754
          using g32 = FractionField<i32>:
02755
02758
          using fpq32 = FractionField<polynomial<q32>>;
02759
02762
          using g64 = FractionField<i64>:
02763
02765
          using pi64 = polynomial<i64>;
02766
02768
          using pq64 = polynomial<q64>;
02769
02771
          using fpg64 = FractionField<polvnomial<g64>>:
02772
02777
          template<typename Ring, typename v1, typename v2>
02778
          using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02779
02786
          template<typename v>
02787
          using embed_int_poly_in_fractions_t =
02788
                   typename Embed<
02789
                       polynomial<typename v::ring_type>,
02790
                       polynomial<FractionField<typename v::ring_type>»::template type<v>;
02791
02795
          template<int64_t p, int64_t q>
          using make_q64_t = typename q64::template simplify_t<
02796
02797
                       typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02798
02802
          template<int32_t p, int32_t q>
02803
          using make_q32_t = typename q32::template simplify_t<
02804
                       typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>»;
02805
02810
          template<typename Ring, typename v1, typename v2>
using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02811
02816
          template<typename Ring, typename v1, typename v2>
02817
          using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02818
02820
          template<>
          struct Embed<a32, a64> {
02821
02824
               template<typename v>
02825
               using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02826
02827
          template<typename Small, typename Large>
struct Embed<polynomial<Small>, polynomial<Large» {</pre>
02831
02832
02833
           private:
02834
               template<typename v, typename i>
02835
               struct at_low;
02836
02837
               template<typename v, size_t i>
02838
               struct at_index {
02839
                   using type = typename Embed<Small, Large>::template
```

```
type<typename v::template coeff_at_t<i>>>;
02840
02841
02842
              template<typename v, size_t... Is>
02843
              struct at_low<v, std::index_sequence<Is...» {
    using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
02844
02845
02846
02847
           public:
02850
              template<typename v>
             using type = typename at_low<v, typename internal::make_index_sequence_reverse<v::degree +
02851
     1»::type;
02852
         };
02853
02857
          template<typename Ring, auto... xs>
02858
          using make_int_polynomial_t = typename polynomial<Ring>::template val<</pre>
02859
                  typename Ring::template inject_constant_t<xs>...>;
02860
02864
          template<typename Ring, auto... xs>
02865
         using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<
02866
                  typename FractionField<Ring>::template inject_constant_t<xs>...>;
02867 } // namespace aerobus
02868
02869 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02870 namespace aerobus {
02871
       namespace internal {
02872
             template<typename T, size_t x, typename E = void>
02873
             struct factorial {};
02874
02875
              template<typename T, size_t x>
02876
              struct factorial<T, x, std::enable_if_t<(x > 0)» {
02877
              private:
02878
                 template<typename, size_t, typename>
02879
                  friend struct factorial;
              public:
02880
02881
                 using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
     x - 1>::type>;
02882
                 static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02883
02884
02885
              template<typename T>
02886
              struct factorial<T. 0> {
02887
              public:
02888
                 using type = typename T::one;
02889
                  static constexpr typename T::inner_type value = type::template get<typename</pre>
     T::inner_type>();
02890
              };
          } // namespace internal
02891
02892
02896
          template<typename T, size_t i>
02897
          using factorial_t = typename internal::factorial<T, i>::type;
02898
02902
          template<typename T, size_t i>
02903
          inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02904
02905
          namespace internal {
02906
              template<typename T, size_t k, size_t n, typename E = void>
02907
              struct combination_helper {};
02908
02909
              template<typename T, size t k, size t n>
02910
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
02911
                  using type = typename FractionField<T>::template mul_t<</pre>
02912
                      typename combination_helper<T, k - 1, n - 1>::type,
02913
                      makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02914
              };
02915
02916
              template<typename T, size_t k, size_t n>
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)» {
02917
02918
                 using type = typename combination_helper<T, n - k, n>::type;
02919
02920
02921
              template<typename T, size_t n>
02922
              struct combination_helper<T, 0, n> {
                 using type = typename FractionField<T>::one;
02923
02924
02925
02926
              template<typename T, size_t k, size_t n>
02927
              struct combination {
                  using type = typename internal::combination_helper<T, k, n>::type::x;
02928
02929
                  static constexpr typename T::inner_type value =
02930
                              internal::combination_helper<T, k, n>::type::template get<typename</pre>
     T::inner_type>();
02931
02932
          } // namespace internal
02933
02936
          template<tvpename T, size t k, size t n>
```

```
using combination_t = typename internal::combination<T, k, n>::type;
02938
02943
          template<typename T, size_t k, size_t n>
02944
          inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02945
02946
          namespace internal {
02947
               template<typename T, size_t m>
02948
               struct bernoulli;
02949
02950
               template<typename T, typename accum, size_t k, size_t m>
               struct bernoulli_helper {
02951
                  using type = typename bernoulli_helper<
02952
02953
                       addfractions_t<T,
02954
02955
                           accum,
02956
                           mulfractions_t<T,</pre>
02957
                               makefraction_t<T,
                                   combination_t<T, k, m + 1>,
02958
                                    typename T::one>,
02959
02960
                               typename bernoulli<T, k>::type
02961
02962
                       k + 1.
02963
02964
                       m>::type;
02965
               };
02966
02967
               template<typename T, typename accum, size_t m>
02968
               struct bernoulli_helper<T, accum, m, m> {
02969
                   using type = accum;
02970
02971
02972
02973
02974
               template<typename T, size_t m>
02975
               struct bernoulli {
                   using type = typename FractionField<T>::template mul t<</pre>
02976
02977
                       typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02978
                       makefraction_t<T,</pre>
02979
                       typename T::template val<static_cast<typename T::inner_type>(-1)>,
02980
                       typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02981
02982
                   >:
02983
02984
                   template<typename floatType>
02985
                   static constexpr floatType value = type::template get<floatType>();
02986
02987
02988
               {\tt template}{<}{\tt typename}\ {\tt T}{>}
               struct bernoulli<T, 0> {
02989
                  using type = typename FractionField<T>::one;
02990
02991
02992
                   template<typename floatType>
02993
                   static constexpr floatType value = type::template get<floatType>();
02994
          };
} // namespace internal
02995
02996
03000
          template<typename T, size_t n>
03001
          using bernoulli_t = typename internal::bernoulli<T, n>::type;
03002
          template<typename FloatType, typename T, size_t n >
inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
03007
03008
03009
03010
          // bell numbers
03011
          namespace internal {
03012
               template<typename T, size_t n, typename E = void>
03013
               struct bell_helper;
03014
03015
               template <typename T, size_t n>
               struct bell_helper<T, n, std::enable_if_t<(n > 1)» {
03016
03017
                   template<typename accum, size_t i, size_t stop>
03018
                   struct sum_helper {
                   private:
03019
03020
                       using left = typename T::template mul_t<
                                    combination_t<T, i, n-1>,
typename bell_helper<T, i>::type>;
03021
03022
03023
                       using new_accum = typename T::template add_t<accum, left>;
03024
                    public:
03025
                       using type = typename sum_helper<new_accum, i+1, stop>::type;
03026
                   };
03027
03028
                   template<typename accum, size_t stop>
                   struct sum_helper<accum, stop, stop> {
03029
03030
                       using type = accum;
03031
03032
03033
                   using type = typename sum_helper<typename T::zero, 0, n>::type;
03034
               };
```

```
template<typename T>
03036
03037
              struct bell_helper<T, 0> {
03038
                 using type = typename T::one;
03039
03040
03041
              template<typename T>
03042
              struct bell_helper<T, 1> {
03043
                 using type = typename T::one;
03044
03045
          } // namespace internal
03046
03050
          template<typename T, size_t n>
          using bell_t = typename internal::bell_helper<T, n>::type;
03051
03052
          template<typename T, size_t n>
static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
03056
03057
03058
03059
          namespace internal {
              template<typename T, int k, typename E = void>
03060
03061
              struct alternate {};
03062
03063
              template<typename T, int k>
              struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
03064
03065
                 using type = typename T::one;
                   static constexpr typename T::inner_type value = type::template get<typename
03066
     T::inner_type>();
03067
03068
              template<typename T, int k> struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
03069
03070
03071
                  using type = typename T::template sub_t<typename T::zero, typename T::one>;
                   static constexpr typename T::inner_type value = type::template get<typename
03072
     T::inner_type>();
03073
          };
} // namespace internal
03074
03075
          template<typename T, int k>
03079
          using alternate_t = typename internal::alternate<T, k>::type;
03080
03083
          template<typename T, size_t k>
03084
          inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
03085
03086
          namespace internal {
              template<typename T, int n, int k, typename E = void>
03087
03088
               struct stirling_1_helper {};
03089
03090
              {\tt template}{<}{\tt typename}\ {\tt T}{>}
              struct stirling_1_helper<T, 0, 0> {
03091
                  using type = typename T::one;
03092
03093
              };
03094
03095
              template<typename T, int n>
              struct stirling_1_helper<T, n, 0, std::enable_if_t<(n > 0)» {
03096
03097
                  using type = typename T::zero;
03098
              };
03099
03100
              template<typename T, int n>
03101
              struct stirling_1_helper<T, 0, n, std::enable_if_t<(n > 0)» {
03102
                  using type = typename T::zero;
03103
03104
03105
              template<typename T, int n, int k>
03106
              struct stirling_1_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)» {
03107
                   using type = typename T::template sub_t<</pre>
03108
                                    typename stirling_1_helper<T, n-1, k-1>::type,
03109
                                    typename T::template mul_t<</pre>
                                        typename T::template inject constant t<n-1>,
03110
03111
                                        typename stirling_1_helper<T, n-1, k>::type
03112
03113
          } // namespace internal
03114
03115
03120
          template<typename T, int n, int k>
          using stirling_1_signed_t = typename internal::stirling_1_helper<T, n, k>::type;
03121
03122
03127
          template<typename T, int n, int k>
03128
          using stirling_1_unsigned_t = abs_t<typename internal::stirling_1_helper<T, n, k>::type>;
03129
          template<typename T, int n, int k>
static constexpr typename T::inner_type stirling_1_unsigned_v = stirling_1_unsigned_t<T, n, k>::v;
03134
03135
03136
03141
          template<typename T, int n, int k>
03142
          static constexpr typename T::inner_type stirling_1_signed_v = stirling_1_signed_t<T, n, k>::v;
03143
0.3144
          namespace internal {
03145
              template<typename T, int n, int k, typename E = void>
```

```
03146
              struct stirling_2_helper {};
03147
03148
               template<typename T, int n>
03149
               struct stirling_2_helper<T, n, n, std::enable_if_t<(n >= 0)> {
0.3150
                  using type = typename T::one;
03151
               };
03152
03153
               template<typename T, int n>
03154
              struct stirling_2_helper<T, n, 0, std::enable_if_t<(n > 0)» {
03155
                   using type = typename T::zero;
03156
              };
03157
03158
               template<typename T, int n>
03159
              struct stirling_2_helper<T, 0, n, std::enable_if_t<(n > 0)» {
03160
                  using type = typename T::zero;
03161
03162
               template<typename T, int n, int k>
03163
               struct stirling_2_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0) && (k < n)» {
03164
                  using type = typename T::template add_t<
03165
03166
                                    typename stirling_2_helper<T, n-1, k-1>::type,
03167
                                    typename T::template mul_t<
03168
                                         typename T::template inject_constant_t<k>,
03169
                                        typename stirling_2_helper<T, n-1, k>::type
03170
03171
               };
03172
           } // namespace internal
03173
03178
          template<typename T, int n, int k>
          using stirling_2_t = typename internal::stirling_2_helper<T, n, k>::type;
03179
03180
03185
          template<typename T, int n, int k>
03186
          static constexpr typename T::inner_type stirling_2_v = stirling_2_t<T, n, k>::v;
03187
03188
          namespace internal {
               template<typename T>
03189
03190
               struct pow scalar {
03191
                   template<size_t p>
03192
                   static constexpr DEVICE INLINED T func(const T& x) { return p == 0 ? static_cast<T>(1) :
03193
                      p % 2 == 0? func<p/2>(x) * func<<math>p/2>(x):
03194
                        x * func<p/2>(x) * func<p/2>(x);
03195
                   }
03196
               }:
03197
03198
               template<typename T, typename p, size_t n, typename E = void>
03199
               requires IsEuclideanDomain<T>
03200
               struct pow;
03201
               template<typename T, typename p, size_t n>
struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)» {
    using type = typename T::template mul_t
03202
03203
03204
03205
                       typename pow<T, p, n/2>::type,
03206
                       typename pow<T, p, n/2>::type
03207
                  >;
03208
              };
03209
03210
               template<typename T, typename p, size_t n>
               struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)» {
    using type = typename T::template mul_t<
03211
03212
03213
03214
                       typename T::template mul t<
                           typename pow<T, p, n/2>::type,
typename pow<T, p, n/2>::type
03215
03216
03217
03218
03219
              };
03220
03221
               template<typename T, typename p, size_t n>
03222
               struct pow<T, p, n, std::enable_if_t<n == 0» { using type = typename T::one; };</pre>
          } // namespace internal
03223
03224
03229
          template<typename T, typename p, size_t n
03230
          using pow_t = typename internal::pow<T, p, n>::type;
03231
03236
          template<typename T, typename p, size_t n>
03237
          static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
03238
03239
          template<typename T, size_t p>
          static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
03240
      internal::pow scalar<T>::template func(x); }
03241
03242
          namespace internal {
03243
               template<typename, template<typename, size_t> typename, class>
03244
               struct make_taylor_impl;
03245
03246
               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...» {
03247
```

```
using type = typename polynomial<FractionField<T»::template val<typename coeff_at<T,
     Is>::type...>;
            };
03249
03250
03251
03256
          template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
          using taylor = typename internal::make_taylor_impl<</pre>
03258
03259
              coeff_at,
03260
              internal::make_index_sequence_reverse<deg + 1>>::type;
03261
03262
          namespace internal {
03263
              template<typename T, size_t i>
03264
              struct exp_coeff {
03265
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03266
03267
03268
              template<typename T, size_t i, typename E = void>
03269
              struct sin_coeff_helper {};
03270
03271
              template<typename T, size_t i>
03272
              struct sin\_coeff\_helper<T, i, std::enable\_if\_t<(i & 1) == 0» {
                 using type = typename FractionField<T>::zero;
03273
03274
03275
03276
              template<typename T, size_t i>
03277
              struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03278
                  using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03279
03280
03281
              template<typename T, size_t i>
03282
              struct sin_coeff {
03283
                 using type = typename sin_coeff_helper<T, i>::type;
03284
03285
              template<typename T, size_t i, typename E = void>
03286
03287
              struct sh_coeff_helper {};
03289
              template<typename T, size_t i>
03290
              struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03291
                  using type = typename FractionField<T>::zero;
03292
              };
03293
03294
              template<typename T, size_t i>
03295
              struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03296
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03297
03298
03299
              template<typename T, size_t i>
03300
              struct sh coeff {
03301
                  using type = typename sh_coeff_helper<T, i>::type;
03302
03303
03304
              template<typename T, size_t i, typename E = void>
03305
              struct cos_coeff_helper {};
03306
              template<typename T, size_t i>
03308
              struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03309
                  using type = typename FractionField<T>::zero;
03310
0.3.31.1
              template<typename T, size_t i>
struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03312
03313
03314
                 using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03315
03316
03317
              template<typename T, size_t i>
03318
              struct cos coeff {
03319
                  using type = typename cos_coeff_helper<T, i>::type;
03320
03321
03322
              template<typename T, size_t i, typename E = void>
03323
              struct cosh_coeff_helper {};
03324
03325
              template<typename T, size_t i>
03326
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
                  using type = typename FractionField<T>::zero;
03327
03328
03329
03330
              template<typename T, size t i>
              struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03331
03332
                  using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03333
03334
03335
              template<typename T, size_t i>
03336
              struct cosh coeff {
                  using type = typename cosh_coeff_helper<T, i>::type;
03337
```

```
03338
              };
03339
03340
              template<typename T, size_t i>
03341
              struct geom_coeff { using type = typename FractionField<T>::one; };
03342
03343
03344
              template<typename T, size_t i, typename E = void>
03345
              struct atan_coeff_helper;
03346
              template<typename T, size_t i>
03347
              struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03348
                 using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>»;
03349
03350
03351
03352
              template<typename T, size_t i>
03353
              using type = typename FractionField<T>::zero;
03354
03355
03356
03357
              template<typename T, size_t i>
03358
              struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
03359
03360
              template<typename T, size_t i, typename E = void>
03361
              struct asin coeff helper;
03362
03363
              template<typename T, size_t i>
03364
              struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03365
                  using type = makefraction_t<T,
03366
                      factorial_t<T, i - 1>,
                      typename T::template mul_t<</pre>
03367
                          typename T::template val<i>,
03368
03369
                          T::template mul_t<
03370
                              pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
03371
                              pow<T, factorial_t<T, i / 2>, 2
03372
03373
03374
                      »;
03375
              };
03376
03377
              template<typename T, size_t i>
03378
              struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0\times {
03379
                  using type = typename FractionField<T>::zero;
03380
03381
03382
              template<typename T, size_t i>
03383
              struct asin_coeff {
03384
                 using type = typename asin_coeff_helper<T, i>::type;
03385
              };
03386
03387
              template<typename T, size t i>
03388
              struct lnp1_coeff {
03389
                  using type = makefraction_t<T,
03390
                      alternate_t<T, i + 1>
03391
                      typename T::template val<i>;
03392
              };
03393
03394
              template<typename T>
03395
              struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
03396
03397
              template<typename T, size_t i, typename E = void>
03398
              struct asinh_coeff_helper;
03399
03400
              template<typename T, size_t i>
03401
              struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
03402
                  using type = makefraction_t<T,
03403
                      typename T::template mul_t<</pre>
03404
                          alternate_t<T, i / 2>,
factorial_t<T, i - 1>
03405
03406
03407
                      typename T::template mul_t<</pre>
03408
                          typename T::template mul_t<</pre>
03409
                              typename T::template val<i>,
03410
                              pow_t<T, factorial_t<T, i / 2>, 2>
03411
03412
                          pow t<T, typename T::template inject constant t<4>, i / 2>
03413
03414
03415
03416
03417
              template<typename T, size t i>
              struct asinh coeff helper<T, i, std::enable if t<(i & 1) == 0» {
03418
                  using type = typename FractionField<T>::zero;
03419
03420
03421
03422
              template<typename T, size_t i>
              struct asinh_coeff {
03423
03424
                  using type = typename asinh_coeff_helper<T, i>::type;
```

```
03425
03426
03427
               template<typename T, size_t i, typename E = void>
03428
               struct atanh_coeff_helper;
03429
               template<typename T, size_t i>
03430
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03431
03432
                    // 1/i
03433
                    using type = typename FractionField<T>:: template val<
                        typename T::one.
03434
                        typename T::template inject_constant_t<i>;;
03435
03436
               };
03437
               template<typename T, size_t i>
03438
03439
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
03440
                   using type = typename FractionField<T>::zero;
03441
               };
03442
03443
               template<typename T, size_t i>
03444
               struct atanh_coeff {
                   using type = typename atanh_coeff_helper<T, i>::type;
03445
03446
03447
               template<typename T, size_t i, typename E = void>
03448
03449
               struct tan_coeff_helper;
03450
               template<typename T, size_t i>
03451
               struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
    using type = typename FractionField<T>::zero;
03452
03453
03454
03455
03456
               template<typename T, size_t i>
03457
               struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
03458
               private:
                    // 4^((i+1)/2)
03459
                    using _4p = typename FractionField<T>::template inject_t<</pre>
03460
                        pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
03461
                    // 4^((i+1)/2) - 1
03462
03463
                    using _4pm1 = typename FractionField<T>::template
      sub_t<_4p, typename FractionField<T>::one>;
    // (-1)^((i-1)/2)
03464
                   using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»; using dividend = typename FractionField<T>::template mul_t<
03465
03466
03467
                        altp,
                        FractionField<T>::template mul_t<
03468
03469
                        -4p,
03470
                        FractionField<T>::template mul_t<
03471
                         _4pm1,
                        bernoulli_t<T, (i + 1)>
03472
03473
03474
03475
               public:
03476
03477
                   using type = typename FractionField<T>::template div_t<dividend,</pre>
03478
                        typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03479
               };
03480
03481
               template<typename T, size_t i>
03482
               struct tan_coeff {
03483
                   using type = typename tan_coeff_helper<T, i>::type;
03484
03485
03486
               template<typename T, size_t i, typename E = void>
03487
               struct tanh_coeff_helper;
03488
03489
               template<typename T, size_t i>
               struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
    using type = typename FractionField<T>::zero;
03490
03491
03492
03493
03494
               template<typename T, size_t i>
03495
               struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
               private:
03496
                   using _4p = typename FractionField<T>::template inject_t<</pre>
03497
                   pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
using _4pml = typename FractionField<T>::template
03498
      sub_t<_4p, typename FractionField<T>::one>;
03500
                   using dividend =
03501
                        typename FractionField<T>::template mul_t<</pre>
                             -4p,
03502
03503
                             typename FractionField<T>::template mul_t<</pre>
03504
                                 _4pm1,
03505
                                 bernoulli_t<T, (i + 1) >>::type;
               public:
03506
03507
                   using type = typename FractionField<T>::template div_t<dividend,</pre>
03508
                        FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
03509
               };
```

```
03511
                       template<typename T, size_t i>
03512
                      struct tanh_coeff {
03513
                           using type = typename tanh_coeff_helper<T, i>::type;
03514
03515
                } // namespace internal
03516
03520
                template<typename Integers, size_t deg>
03521
                using exp = taylor<Integers, internal::exp_coeff, deg>;
03522
03526
                template<typename Integers, size_t deg>
                using expm1 = typename polynomial<FractionField<Integers>>::template sub_t<
03527
03528
                      exp<Integers, deg>,
                      typename polynomial<FractionField<Integers>>::one>;
03529
03530
                template<typename Integers, size_t deg>
03534
                using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
03535
03536
03540
                template<typename Integers, size_t deg>
03541
                using atan = taylor<Integers, internal::atan_coeff, deg>;
03542
03546
                template<typename Integers, size_t deg>
03547
                using sin = taylor<Integers, internal::sin_coeff, deg>;
03548
03552
                template<typename Integers, size_t deg>
03553
                using sinh = taylor<Integers, internal::sh_coeff, deg>;
03554
03559
                template<typename Integers, size_t deg>
03560
                using cosh = taylor<Integers, internal::cosh_coeff, deg>;
03561
03566
                template<typename Integers, size_t deg>
03567
                using cos = taylor<Integers, internal::cos_coeff, deg>;
03568
03573
                template<typename Integers, size_t deg>
03574
                using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
03575
03580
               template<typename Integers, size_t deg>
using asin = taylor<Integers, internal::asin_coeff, deg>;
03581
03582
03587
                template<typename Integers, size_t deg>
03588
                using asinh = taylor<Integers, internal::asinh_coeff, deg>;
03589
03594
                template<typename Integers, size_t deg>
03595
                using atanh = taylor<Integers, internal::atanh_coeff, deg>;
03596
03601
                template<typename Integers, size_t deg>
03602
                using tan = taylor<Integers, internal::tan_coeff, deg>;
03603
                template<typename Integers, size t deg>
03608
03609
                using tanh = taylor<Integers, internal::tanh_coeff, deg>;
03610 }
              // namespace aerobus
03611
03612 // continued fractions
03613 namespace aerobus {
               template<int64 t... values>
03616
03617
               struct ContinuedFraction {};
03618
03621
                template<int64 t a0>
03622
                struct ContinuedFraction<a0> {
03624
                      using type = typename q64::template inject_constant_t<a0>;
03626
                      static constexpr double val = static cast<double>(a0);
03627
                };
03628
03632
                template<int64_t a0, int64_t... rest>
                struct ContinuedFraction<a0, rest...> {
03633
03635
                      using type = q64::template add_t<
03636
                                    typename q64::template inject_constant_t<a0>,
03637
                                    typename g64::template div_t<
03638
                                          typename q64::one,
03639
                                          typename ContinuedFraction<rest...>::type
03640
03641
03643
                      static constexpr double val = type::template get<double>();
03644
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>;
03651
               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>;
03653
               using SORT2 fraction =
         03655
               using SQRT3_fraction =
         ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 
          // NOLINT
03656 } // namespace aerobus
03657
03658 // known polynomials
```

```
03659 namespace aerobus {
          // CChebyshev
03661
          namespace internal {
03662
              template<int kind, size_t deg, typename I>
03663
              struct chebyshev_helper {
03664
                  using type = typename polynomial<I>::template sub t<
                      typename polynomial<I>::template mul_t<
03665
03666
                           typename polynomial<I>::template mul_t<</pre>
03667
                              typename polynomial<I>::template inject_constant_t<2>,
03668
                              typename polynomial<I>::X>,
                          typename chebyshev_helper<kind, deg - 1, I>::type
03669
03670
03671
                      typename chebyshev_helper<kind, deg - 2, I>::type
03672
03673
              };
03674
03675
              template<typename I>
03676
              struct chebyshev_helper<1, 0, I> {
03677
                  using type = typename polynomial<I>::one;
03678
              };
03679
03680
              template<typename I>
03681
              struct chebyshev_helper<1, 1, I> \{
03682
                  using type = typename polynomial<I>::X;
03683
03684
03685
              template<typename I>
03686
              struct chebyshev_helper<2, 0, I> {
03687
                  using type = typename polynomial<I>::one;
03688
03689
03690
              template<typename I>
03691
              struct chebyshev_helper<2, 1, I> {
03692
                  using type = typename polynomial<I>::template mul_t<</pre>
                      typename polynomial<I>::template inject_constant_t<2>,
03693
03694
                      typename polynomial<I>::X>;
03695
          } // namespace internal
03696
03697
03698
          // Laguerre
03699
          namespace internal {
              template<size_t deg, typename I>
03700
03701
              struct laquerre helper {
03702
                  using Q = FractionField<I>;
03703
                  using PQ = polynomial<Q>;
03704
03705
                  // Lk = (1 / k) * ((2 * k - 1 - x) * 1km1 - (k - 2)Lkm2)
03706
03707
                  using lnm2 = typename laguerre_helper<deg - 2, I>::type;
                  using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03708
03709
                  // -x + 2k-1
03710
                  using p = typename PQ::template val<
03711
                      typename Q::template inject_constant_t<-1>,
03712
                      typename Q::template inject_constant_t<2 * deg - 1»;</pre>
03713
                  // 1/n
03714
                  using factor = typename PO::template inject ring t<
                      typename Q::template val<typename I::one, typename I::template
      inject_constant_t<deg>>;
03716
03717
               public:
                  using type = typename PQ::template mul_t <</pre>
03718
03719
                      factor,
03720
                      typename PQ::template sub_t<
03721
                           typename PQ::template mul_t<</pre>
03722
03723
                               1 nm1
03724
03725
                           typename PQ::template mul_t<</pre>
03726
                               typename PQ::template inject_constant_t<deg-1>,
03727
03728
03729
03730
                  >;
03731
              };
03732
03733
              template<typename I>
03734
              struct laguerre_helper<0, I> {
03735
                  using type = typename polynomial<FractionField<I>::one;
03736
              };
03737
03738
              template<typename I>
03739
              struct laguerre_helper<1, I> {
              private:
03740
03741
                  using PQ = polynomial<FractionField<I>;
               public:
03742
                  using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
03743
03744
```

```
03745
          } // namespace internal
03746
03747
          // Bernstein
03748
          namespace internal {
03749
             template<size_t i, size_t m, typename I, typename E = void>
03750
              struct bernstein_helper {};
03751
03752
              template<typename I>
03753
              struct bernstein_helper<0, 0, I> {
03754
                  using type = typename polynomial<I>::one;
03755
              };
03756
              03757
03758
03759
               private:
03760
                  using P = polynomial<I>;
03761
03762
               public:
03763
                 using type = typename P::template mul_t<
03764
                          typename P::template sub_t<typename P::one, typename P::X>,
03765
                          typename bernstein_helper<i, m-1, I>::type>;
03766
03767
03768
              template<size_t i, size_t m, typename I>
struct bernstein_helper<i, m, I, std::enable_if_t<</pre>
03769
03770
                         (m > 0) && (i == m)  (
03771
               private:
03772
                  using P = polynomial<I>;
03773
               public:
03774
                  using type = typename P::template mul_t<
03775
                          typename P::X,
03776
                          typename bernstein_helper<i-1, m-1, I>::type>;
03777
03778
03779
              template<size_t i, size_t m, typename I>
              03780
03781
03782
03783
                  using P = polynomial<I>;
03784
               public:
03785
                  using type = typename P::template add_t<
03786
                           typename P::template mul_t<</pre>
03787
                              typename P::template sub_t<typename P::one, typename P::X>,
03788
                               typename bernstein_helper<i, m-1, I>::type>,
03789
                           typename P::template mul_t<
03790
                              typename P::X,
03791
                              typename bernstein_helper<i-1, m-1, I>::type»;
03792
          };
} // namespace internal
03793
03794
03795
          // AllOne polynomials
03796
          namespace internal {
03797
              template<size_t deg, typename I>
03798
              struct AllOneHelper {
03799
                  using type = aerobus::add t<
03800
                      typename polynomial<I>::one,
03801
                      typename aerobus::mul_t<
                           typename polynomial<I>::X,
03802
03803
                           typename AllOneHelper<deg-1, I>::type
03804
03805
              };
03806
03807
              template<typename I>
03808
              struct AllOneHelper<0, I> {
03809
                  using type = typename polynomial<I>::one;
03810
03811
          } // namespace internal
03812
03813
          // Bessel polynomials
          namespace internal {
03814
03815
              template<size_t deg, typename I>
03816
              struct BesselHelper {
               private:
03817
                  using P = polynomial<I>;
03818
                  using factor = typename P::template monomial_t<
typename I::template inject_constant_t<(2*deg - 1)>,
03819
03820
03821
                      1>;
03822
               public:
                  using type = typename P::template add_t<
    typename P::template mul_t<</pre>
03823
03824
03825
                          factor,
03826
                           typename BesselHelper<deg-1, I>::type
03827
03828
                      typename BesselHelper<deg-2, I>::type
03829
                  >;
03830
              };
03831
```

```
template<typename I>
              struct BesselHelper<0, I> {
03833
03834
                  using type = typename polynomial<I>::one;
03835
03836
03837
              template<tvpename I>
03838
              struct BesselHelper<1, I> {
03839
              private:
03840
                  using P = polynomial<I>;
03841
               public:
03842
                  using type = typename P::template add_t<
03843
                     typename P::one,
03844
                      typename P::X
03845
03846
              } ;
03847
         } // namespace internal
03848
03849
         namespace known_polynomials {
03851
              enum hermite_kind {
                 probabilist,
03853
03855
                  physicist
03856
              };
03857
         }
03858
03859
          // hermite
03860
          namespace internal {
03861
              template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03862
              struct hermite_helper {};
03863
03864
              template<size_t deg, typename I>
03865
              struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
03866
              private:
                  using hnm1 = typename hermite_helper<deg - 1,
03867
      known_polynomials::hermite_kind::probabilist, I>::type;
03868
                 using hnm2 = typename hermite_helper<deg - 2,
     known_polynomials::hermite_kind::probabilist, I>::type;
03869
               public:
03870
03871
                  using type = typename polynomial<I>::template sub_t<
03872
                      typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnml>,
03873
                      typename polynomial<I>::template mul_t<</pre>
03874
                          typename polynomial<I>::template inject_constant_t<deg - 1>,
03875
                          hnm2
03876
03877
                  >;
03878
              } ;
03879
03880
              template<size_t deg, typename I>
03881
              struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03882
              private:
03883
                  using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
     I>::type;
03884
                  using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
     I>::type;
03885
03886
               public:
03887
                  using type = typename polynomial<I>::template sub_t<
03888
                       // 2X Hn-1
03889
                      typename polynomial<I>::template mul_t<</pre>
03890
                          typename pi64::val<typename I::template inject_constant_t<2>,
03891
                          typename I::zero>, hnm1>,
03892
03893
                      typename polynomial<I>::template mul_t<</pre>
03894
                          typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03895
                          hnm2
03896
03897
                  >;
03898
              };
03899
03900
              template<typename I>
03901
              struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03902
                  using type = typename polynomial<I>::one;
03903
              };
03904
03905
              template<typename I>
03906
              struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03907
                 using type = typename polynomial<I>::X;
03908
03909
03910
              template<tvpename T>
03911
              struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03912
                  using type = typename pi64::one;
03913
03914
03915
              template<typename I>
03916
              struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03917
                  // 2X
```

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

```
struct Inner {
                           // abel(n, a) = (x-an) * abel(n-1, a)
04006
04007
                           using type = typename aerobus::mul_t<
04008
                                typename Inner<deg-1, an>::type,
04009
                                typename aerobus::sub_t<typename P::X, typename P::template inject_constant_t<an>>
04010
                           >;
04011
                      };
04012
04013
                      // abel(0, a) = 1
04014
                      template<I::inner_type an>
04015
                      struct Inner<0, an> {
04016
                           using type = P::one;
04017
04018
04019
                      // abel(1, a) = X
04020
                      template<I::inner_type an>
                      struct Inner<1, an> {
    using type = P::X;
04021
04022
04023
04024
                 };
04025
            } // namespace internal
04026
04028
            namespace known_polynomials {
04029
04038
                 template<size_t n, auto a, typename I = aerobus::i64>
04039
                 using abel = typename internal::AbelHelper<I>::template Inner<n, a*n>::type;
04040
04048
                 template <size_t deg, typename I = aerobus::i64>
04049
                 using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
04050
04060
                 template <size_t deg, typename I = aerobus::i64>
04061
                 using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
04062
04072
                 template <size_t deg, typename I = aerobus::i64>
04073
                 using laguerre = typename internal::laguerre_helper<deg, I>::type;
04074
                 template <size_t deg, typename I = aerobus::i64>
using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,</pre>
04081
04082
       I>::type;
04083
04090
                 template <size_t deg, typename I = aerobus::i64>
04091
                 using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
04092
04103
                 template<size_t i, size_t m, typename I = aerobus::i64>
                 using bernstein = typename internal::bernstein_helper<i, m, I>::type;
04104
04105
04115
                 template<size_t deg, typename I = aerobus::i64>
04116
                 using legendre = typename internal::legendre_helper<deg, I>::type;
04117
04127
                 template<size t deg, typename I = aerobus::i64>
04128
                 using bernoulli = taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
04129
04136
                 template<size_t deg, typename I = aerobus::i64>
04137
                 using allone = typename internal::AllOneHelper<deg, I>::type;
04138
04146
                 template<size_t deg, typename I = aerobus::i64>
using bessel = typename internal::BesselHelper<deg, I>::type;
04147
04148
04156
                 template<size_t deg, typename I = aerobus::i64>
04157
                 using touchard = taylor<I, internal::touchard_coeff<deg>::template inner, deg>;
04158
               // namespace known_polynomials
04159 } // namespace aerobus
04160
04161
04162 #ifdef AEROBUS_CONWAY_IMPORTS
04163
04164 // conway polynomials
04165 namespace aerobus {
04169
           template<int p, int n>
            struct ConwayPolynomial {};
04171
04172 #ifndef DO_NOT_DOCUMENT
04173
            #define ZPZV ZPZ::template val
            #define POLYV aerobus::polynomial<ZPZ>::template val
04174
            template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
04175
       POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
04176
            template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
       POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
04177
            template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1), ; // NOLINT
template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
04178
            template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
04179
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
       template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>; ysing type =
template<> struct ConwayPolynomial<2, 7> { using ZPZ = aerobus::zpz<2>; using type =
04180
04181
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
                                                              template<> struct ConwayPolynomial<2, 8> { using ZPZ = aerobus::zpz<2>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPŽV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPŽV<1»; };
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           // NOLINT
                                                          template<> struct ConwayPolynomial<2, 9> { using ZPZ = aerobus::zpz<2>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    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 =
04185
                                    POLYY<ZPZV<1>, 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
04186
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
                                      ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                                            template<> struct ConwayPolynomial<2, 13> { using ZPZ = aerobus::zpz<2>; using type =
04187
                                    POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
                                      ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                                               template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>,
                                      ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
04189
                                                              template<> struct ConwayPolynomial<2, 15> { using ZPZ = aerobus::zpz<2>; using type =
                                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04190
                                                              template<> struct ConwayPolynomial<2, 16> { using ZPZ = aerobus::zpz<2>; using type
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<2, 17> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
 04191
                                    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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1»; };</pre>
                                                          template<> struct ConwayPolynomial<2, 19> { using ZPZ = aerobus::zpz<2>; using type =
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
                                      NOLINT
                                    template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                       ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1</pre>; };
                                       // NOLINT
04195
                                                             template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
                                   POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
 04196
                                                             template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
                                  POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2»; }; // NOLINT
 04197
                                                              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 =
 04198
                                  POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; };
                                                                                                                                                                                                                                                                                                                                                                                          // NOLINT
                                                             template<> struct ConwayPolynomial<3, 5> { using ZPZ = aerobus::zpz<3>; using type =
 04199
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT
                                                              template<> struct ConwayPolynomial<3, 6> { using ZPZ = aerobus::zpz<3>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<3, 7> { using ZPZ = aerobus::zpz<3>; using type =
 04201
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<1»; }; // NOLINT
                                                              template<> struct ConwayPolynomial<3, 8> { using ZPZ = aerobus::zpz<3>; using type =
 04202
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>; };
                                                           template<> struct ConwayPolynomial<3, 9> { using ZPZ = aerobus::zpz<3>; using type
 04203
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; //
                                      NOLINT
04204
                                                            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<0>, ZPZV<0>, ZPZV<1>,
                                      ZPZV<2»; }; // NOLINT</pre>
                                                               template<> struct ConwayPolynomial<3, 11> { using ZPZ = aerobus::zpz<3>; using type
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<3, 12> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT
04206
                                                               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
                                                            template<> struct ConwayPolynomial<3, 14> { using ZPZ = aerobus::zpz<3>; using type =
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<1>,
                                    ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZ
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1>; }; // NOLINT
template<> struct ConwayPolynomial<3, 16> { using ZPZ = aerobus::zpz<3>; using type =
                                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2»; }; // NOLINT</pre>
                                                               template<> struct ConwayPolynomial<3, 17> { using ZPZ = aerobus::zpz<3>; using type =
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<3, 18> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<3</pre>
04212
```

```
template<> struct ConwayPolynomial<3, 19> { using ZPZ = aerobus::zpz<3>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1»; };</pre>
                             NOLINT
                             template<> struct ConwayPolynomial<3, 20> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
                              ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
                                                template<> struct ConwayPolynomial<5, 1> { using ZPZ = aerobus::zpz<5>; using type =
04215
                            POLYV<ZPZV<1>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<5, 2> { using ZPZ = aerobus::zpz<5>; using type =
04216
                            POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2»; }; // NOLINT
04217
                                                template<> struct ConwayPolynomial<5, 3> { using ZPZ = aerobus::zpz<5>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3»; }; // NOLINT
04218
                                              template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<5, 5> { using ZPZ = aerobus::zpz<5>; using type =
04219
                           POLYV<ZPZV<1>, 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 =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3»; };
                                                                                                                                                                                                                                                                                                                                                                                                                                         // NOLINT
04222
                                              template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<3, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<5, 9> { using ZPZ = aerobus::zpz<5>; using type =
04223
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<3»; }; //
                            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<4>,
04224
                             ZPZV<2»: }: // NOLINT</pre>
                            template<> struct ConwayPolynomial<5, 11> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                             template<> struct ConwayPolynomial<5, 12> { using ZPZ = aerobus::zpz<5>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<2>; }; // NOLINT
                                                template<> struct ConwayPolynomial<5, 13> { using ZPZ = aerobus::zpz<5>; using type =
04227
                              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 =
                            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5, ZPZV<5,
04229
                             ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<3»; }; // NOLINT</pre>
04230
                                                template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4 , ZPZV<5 , ZPZV<6 , ZPZV<6
04231
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZ
                                               template<> struct ConwayPolynomial<5, 18> { using ZPZ = aerobus::zpz<5>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                            ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<5, 19> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
04233
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2»; }; //</pre>
                            template<> struct ConwayPolynomial<5, 20> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<2>; };
                             // NOLINT
04235
                                                 template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
                            POLYV<ZPZV<1>, ZPZV<4»; }; // NOLINT
04236
                                               template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
                            POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3»; }; // NOLINT
04237
                                                template<> struct ConwayPolynomial<7, 3> { using ZPZ = aerobus::zpz<7>; using type =
                           POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4»; }; // NOLINT template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type =
04238
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
04239
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT
                                                template<> struct ConwayPolynomial<7, 6> { using ZPZ = aerobus::zpz<7>; using type =
04240
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<4>, ZPZV<6>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<7, 7> { using ZPZ = aerobus::zpz<7>; using type
04241
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<4»; }; // NOLINT
                                               template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
04242
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<6>, ZPZV<6>, ZPZV<2>, ZPZV<3»; }; // NOLINT
04243
                                               template<> struct ConwayPolynomial<7, 9> { using ZPZ = aerobus::zpz<7>; using type =
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                             NOLINT
                                                template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, 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>, ZPZV<1>, ZPZV<4»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<7, 12> { using ZPZ = aerobus::zpz<7>; using type
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<5>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5, ZPZV<
                                          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»; }; // NOLINT</pre>
                                                                        template<> struct ConwayPolynomial<7, 14> { using ZPZ = aerobus::zpz<7>; using type
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<6>,
                                             ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<3»; }; // NOLINT</pre>
                                                                       template<> struct ConwayPolynomial<7, 15> { using ZPZ = aerobus::zpz<7>; using type =
 04249
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           ZPZV<6>, ZPZV<6>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<4»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<7, 165 { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>,
 04250
                                           ZPZV<3>, ZPZV<4>, ZPZV<1>, ZPZV<6>, ZPZV<2>, ZPZV<4>, ZPZV<3»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<7, 17> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
 04251
                                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>; // NOLINT 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<1>,
                                            \texttt{ZPZV} < 6>, \ \texttt{ZPZV} < 5>, \ \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 3>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 6>, \ \texttt{ZPZV} < 2>, \ \texttt{ZPZV} < 2>, \ \texttt{ZPZV} < 3 »; \ \texttt{}// \ \texttt{NOLINT} 
 04253
                                                                       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<5>, ZPZV<4»; }; //</pre>
                                           NOLINT
                                          template<> struct ConwayPolynomial<7, 20> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                             ZPZV<2>, ZPZV<5>, ZPZV<2>, ZPZV<3>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<3>, ZPZV<3>
                                             // NOLINT
 04255
                                                                       template<> struct ConwayPolynomial<11, 1> { using ZPZ = aerobus::zpz<11>; using type =
                                          POLYV<ZPZV<1>, ZPZV<9»; }; // NOLINT
 04256
                                                                        template<> struct ConwayPolynomial<11, 2> { using ZPZ = aerobus::zpz<11>; using type =
                                           POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2»; }; // NOLINT
  04257
                                                                  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 =
  04258
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<11, 5> { using ZPZ = aerobus::zpz<11>; using type =
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<0>, ZPZV<9»; }; // NOLINT
                                                                     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<0>, ZPZV<4>, ZPZV<9»; }; // NOLINT
 04261
 04262
                                                                      template<> struct ConwayPolynomial<11, 8> { using ZPZ = aerobus::zpz<11>; using type =
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV×7>, ZPZV×7
  04263
                                                                       template<> struct ConwayPolynomial<11, 9> { using ZPZ = aerobus::zpz<11>; using type
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9
                                           NOLINT
 04264
                                                                     template<> struct ConwayPolynomial<11, 10> { using ZPZ = aerobus::zpz<11>; using type =
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 ), 
                                           ZPZV<2»; }; // NOLINT
                                                                      template<> struct ConwayPolynomial<11, 11> { 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<10>, ZPZV<9»; }; // NOLINT

template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<2»; }; // NOLINT
 04266
                                                                   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 ,
04268
                                          ZPZV<4>, ZPZV<6>, ZPZV<6>, ZPZV<10>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<11, 15> { 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<10>,
                                           ZPZV<7>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<9»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<11, 16> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<10>, ZPZV<2>; }; // NOLINT
                                                                          template<> struct ConwayPolynomial<11, 17> { using ZPZ = aerobus::zpz<11>; using type
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<9»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<11, 18> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<8>, ZPZV<8-, ZPZV<8-,
 04272
                                          ZPZV<3>, ZPZV<9>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<9>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT
template<> struct ConwayPolynomial<11, 19> { using ZPZ = aerobus::zpz<11>; using type =
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<2>, ZPZV<2>, ZPZV<3</pre>; };
                                           NOLINT
 04274
                                          template<> struct ConwayPolynomial<11, 20> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</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
  04275
                                                                      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 =
  04276
                                           POLYV<ZPZV<1>, ZPZV<12>, ZPZV<2»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<13, 3> { using ZPZ = aerobus::zpz<13>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<13, 4> { using ZPZ = aerobus::zpz<13>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<13, 5> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<11»; }; // NOLINT
04279
                                                    template<> struct ConwayPolynomial<13, 6> { using ZPZ = aerobus::zpz<13>; using type =
                               POLYV<ZPZV<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 =
04281
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<11»; };
04282
                                                   template<> struct ConwayPolynomial<13, 8> { using ZPZ = aerobus::zpz<13>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3>; }; template<> struct ConwayPolynomial<13, 9> { using ZPZ = aerobus::zpz<13>; using type =
04283
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<
                                // NOLINT
                                                   template<> struct ConwayPolynomial<13, 10> { using ZPZ = aerobus::zpz<13>; using type =
04284
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>,
                                ZPZV<2»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<13, 11> { using ZPZ = aerobus::zpz<13>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<3>, ZPZV<11»; };</pre>
                                                                                                                                                      // NOLINT
04286
                                                    template<> struct ConwayPolynomial<13, 12> { using ZPZ = aerobus::zpz<13>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<8>, ZPZV<11>, ZPZV<3>, ZPZV<1>, ZPZV<4>, ZPZV<4 , ZPZV<
04287
                                                    template<> struct ConwayPolynomial<13, 13> { using ZPZ = aerobus::zpz<13>; using type
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                 template<> struct ConwayPolynomial<13, 14> { using ZPZ = aerobus::zpz<13>; using type
                               POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<5>, ZPZV<5
                               ZPZV<11>, ZPZV<7>, ZPZV<10>, ZPZV<10>, ZPZV<20>; }; // NOLINT template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>,
                                ZPZV<2>, ZPZV<11>, ZPZV<10>, ZPZV<11>, ZPZV<8>, ZPZV<11»; // NOLINT template<> struct ConwayPolynomial<13, 16> { using ZPZ = aerobus::zpz<13>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<8>, ZPZV<2>, ZPZV<2>, ZPZV<12>, ZPZV<12>, ZPZV<6>, ZPZV<6>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<13, 17> { using ZPZ = aerobus::zpz<13>; using type =
04291
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0
                                                  template<> struct ConwayPolynomial<13, 18> { using ZPZ = aerobus::zpz<13>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<4>,
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          7.PZV<11>.
                               ZPZV<11>, ZPZV<9>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<0>, ZPZV<9, ZPZV<2; }; // NOLINT template<> struct ConwayPolynomial<13, 19> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
04293
                                ZPZV<0>, ZPZV<9>, ZPZV<9</pre>
04294
                                                  template<> struct ConwayPolynomial<13, 20> { using ZPZ = aerobus::zpz<13>; using type
                               POLYYCZPZVC1>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC1>, ZPZVC1>, ZPZVC1>, ZPZVC2>, ZPZVC0>, ZPZVC1>, ZPZVC2>, ZPZVC0>, ZPZVC0>, ZPZVC1>, ZPZVC2>, ZPZVC0>, ZPZVC0>, ZPZVC1>, ZPZVC2>, ZPZVC2>, ZPZVC2>, ZPZVC0>, ZPZVC1>, ZPZVC2>, ZPZVC2
                                // NOLINT
04295
                                                     template<> struct ConwayPolynomial<17, 1> { using ZPZ = aerobus::zpz<17>; using type =
                               POLYV<ZPZV<1>, ZPZV<14»; }; // NOLINT
04296
                                                  template<> struct ConwayPolynomial<17, 2> { using ZPZ = aerobus::zpz<17>; using type =
                               POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
04297
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<10>, ZPZV<3»; }; // NOLINT
04299
                                                     template<> struct ConwayPolynomial<17, 5> { using ZPZ = aerobus::zpz<17>; using type =
                               template<> struct ConwayPolynomial<17, 6> { using ZPZ = aerobus::zpz<17>; using type =
04300
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<10>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>; }; // NoLINT template<> struct ConwayPolynomial<17, 7> { using ZPZ = aerobus::zpz<17>; using type
04301
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<14»; }; // NOLINT
04302
                                                 template<> struct ConwayPolynomial<17, 8> { using ZPZ = aerobus::zpz<17>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<12>, ZPZV<0>, ZPZV<6>, ZPZV<3»; };
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      // NOLINT
                               template<> struct ConwayPolynomial<17, 9> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<84»; };</pre>
04303
                                                     template<> struct ConwayPolynomial<17, 10> { using ZPZ = aerobus::zpz<17>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<12>,
                                ZPZV<3»; }; // NOLINT</pre>
                               \label{template} $$ \text{template}$ > \text{struct ConwayPolynomial} < 17, 11> \{ using ZPZ = aerobus:: zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04305
                                                                                                                                                        // NOLINT
                                ZPZV<5>, ZPZV<14»; };</pre>
                                                    template<> struct ConwayPolynomial<17, 12> { using ZPZ = aerobus::zpz<17>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<4>, ZPZV<44, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<16>,
                                ZPZV<14>, ZPZV<9>, ZPZV<3»; };</pre>
                                                                                                                                                                                                     // NOLINT
                                                    template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type =
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<6>,
                                ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
04309
                                                  template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::zpz<17>; using type =
                               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
```

```
template<> struct ConwayPolynomial<17, 16> { using ZPZ = aerobus::zpz<17>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
                          ZPZV<5>, ZPZV<2>, ZPZV<12>, ZPZV<13>, ZPZV<12>, ZPZV<1>, ZPZV<3»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<17, 17> { using ZPZ = aerobus::zpz<17>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1*; ; // NOLINT template<> struct ConwayPolynomial<17, 18> { using ZPZ = aerobus::zpz<17>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<16>,
                         ZPZV<7>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<91>, ZPZV<13>, ZPZV<13>, ZPZV<9>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<17, 19> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
04313
                          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<5>,
                          ZPZV<16>, ZPZV<14>, ZPZV<13>, ZPZV<3>, ZPZV<14>, ZPZV<9>, ZPZV<1>, ZPZV<13>, ZPZV<2>, ZPZV<5>,
                          ZPZV<3»: }: // NOLINT
04315
                                          template<> struct ConwayPolynomial<19, 1> { using ZPZ = aerobus::zpz<19>; using type =
                          POLYV<ZPZV<1>, ZPZV<17»; }; // NOLINT
                                          template<> struct ConwayPolynomial<19, 2> { using ZPZ = aerobus::zpz<19>; using type =
                          POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2»; }; // NOLINT
 04317
                                           template<> struct ConwayPolynomial<19, 3> { using ZPZ = aerobus::zpz<19>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17»; }; // NOLINT

template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2»; }; // NOLINT
 04318
                                           template<> struct ConwayPolynomial<19, 5> { using ZPZ = aerobus::zpz<19>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17»; }; // NOLINT
 04320
                                          template<> struct ConwayPolynomial<19, 6> { using ZPZ = aerobus::zpz<19>; using type =
                          \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 17>, \ \texttt{ZPZV} < 6>, \ \texttt{ZPZV} < 2>; \ \}; \ \ // \ \texttt{NOLINTERMATION } 
                                        template<> struct ConwayPolynomial<19, 7> { using ZPZ = aerobus::zpz<19>; using type =
04321
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<17»; }; // NOLINT
 04322
                                           template<> struct ConwayPolynomial<19, 8> { using ZPZ = aerobus::zpz<19>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<10>, ZPZV<3>, ZPZV<2»; };
                                       template<> struct ConwayPolynomial<19, 9> { using ZPZ = aerobus::zpz<19>; using type
 04323
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<16>, ZPZV<17»; };
                          // NOLINT
                                          template<> struct ConwayPolynomial<19, 10> { using ZPZ = aerobus::zpz<19>; using type =
04324
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>,
                          ZPZV<2»: }: // NOLINT
                                          template<> struct ConwayPolynomial<19, 11> { using ZPZ = aerobus::zpz<19>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<8>, ZPZV<17»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<19, 12> { using ZPZ = aerobus::zpz<19>; using type
04326
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3
04327
                                           template<> struct ConwayPolynomial<19, 13> { using ZPZ = aerobus::zpz<19>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<17»; }; // NOLINT
template<> struct ConwayPolynomial<19, 14> { using ZPZ = aerobus::zpz<19>; using type =
04328
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<11>, ZPZV<11>, ZPZV<11>,
                          ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<7>, ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<19, 15> { using ZPZ = aerobus::zpz<19>; using type =
 04329
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>,
                          ZPZV<11>, ZPZV<13>, ZPZV<15>, ZPZV<14>, ZPZV<0>, ZPZV<17»; }; // NOLINT
template<> struct ConwayPolynomial<19, 16> { using ZPZ = aerobus::zpz<19>; using type =
04330
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<13>, ZPZV<0>, ZPZV<15>, ZPZV<9>, ZPZV<6>, ZPZV<14>, ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<19, 17> { using ZPZ = aerobus::zpz<19>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<17»; };</pre>
                                                                                                                                                                                                                                                                                                                                                       // NOLINT
                                         template<> struct ConwayPolynomial<19, 18> { using ZPZ = aerobus::zpz<19>; using type =
04332
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7
                          ZPZV<17>, ZPZV<5>, ZPZV<0>, ZPZV<16>, ZPZV<5>, ZPZV<7>, ZPZV<3>, ZPZV<14>, ZPZV<2»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<19, 19> { using ZPZ = aerobus::zpz<19>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<17»; }; //</pre>
                          NOLINT
                                         template<> struct ConwayPolynomial<19, 20> { using ZPZ = aerobus::zpz<19>; using type =
04334
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<13>, ZPZV<0>, ZPZV<4>, ZPZV<7>, ZPZV<8>, ZPZV<6>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<6>, ZPZV<11>, ZPZV<2»;
                          }; // NOLINT
                                         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 =
04336
                         POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; }; // NOLINT
                                           template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; }; // NOLINT template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
 04338
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT
04339
                                          template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<18»; }; // NOLINT
                                           template<> struct ConwayPolynomial<23, 6> { using ZPZ = aerobus::zpz<23>; using type =
 04340
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<5>, ZPZV<5>; }; // NOLINT
 04341
                                        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<0>, ZPZV<21>, ZPZV<18»; }; // NOLINT template<> struct ConwayPolynomial<23, 8> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<20>, ZPZV<5>, ZPZV<5>, ZPZV<5>; // NOLINT
 04342
```

```
template<> struct ConwayPolynomial<23, 9> { using ZPZ = aerobus::zpz<23>; using type
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<9>, ZPZV<18»; };
                                           // NOLINT
                                        template<> struct ConwayPolynomial<23, 10> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<15>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6 , ZPZV<6 
  04344
                                          ZPZV<5»: }: // NOLINT
                                                                    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<14>, ZPZV<12>, ZPZV<15>; }; // NOLINT
 04346
                                                                    template<> struct ConwayPolynomial<23, 13> { using ZPZ = aerobus::zpz<23>; using type
 04347
                                          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 =
 04348
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<5>, ZPZV<1>,
                                        ZPZV<18, ZPZV<19, ZPZV<19, ZPZV<22, ZPZV<28, ZPZV<38, ZPZV<19, ZPZV<19, ZPZV<22, ZPZV<38, }; // 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 =
 04350
                                        POLYYCZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
 04351
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<18»; };</pre>
 04352
                                                                template<> struct ConwayPolynomial<23, 18> { using ZPZ = aerobus::zpz<23>; using type =
                                       Template(> Struct ConwayFolynomial<23, 16> ( using ZFZ = aerobus::zpz<25); using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<18, ZPZV<28, ZPZV<2>, ZPZV<1>, ZPZV<18>, ZPZV<3>, ZPZV<16>, ZPZV<15, ZPZV<10>, ZPZV<10>, ZPZV<0>, ZPZV<0 , ZPZV
                                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<18»; }; //</pre>
 04354
                                                                    template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type =
                                        POLYV<ZPZV<1>, ZPZV<27»; }; // NOLINT
                                                                   template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
 04355
                                        POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; }; // NOLINT
                                                                     template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ; ; // NOLINT template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
  04357
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<2»; }; // NOLINT

template<> struct ConwayPolynomial<29, 5> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<27»; }; // NOLINT
  04358
                                                                     template<> struct ConwayPolynomial<29, 6> { using ZPZ = aerobus::zpz<29>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<17>, ZPZV<13>, ZPZV<2»; }; // NOLINT
  04360
                                                               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>, ZPZV<2>, ZPZV<2>; is in type =
  04361
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<2*; };
                                                                    template<> struct ConwayPolynomial<29, 9> { using ZPZ = aerobus::zpz<29>; using type
  04362
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<22>, ZPZV<22>, ZPZV<27»; };
                                            // NOLINT
 04363
                                                                  template<> struct ConwayPolynomial<29, 10> { using ZPZ = aerobus::zpz<29>; using type =
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2
                                           ZPZV<2»; }; // NOLINT</pre>
                                                                template<> struct ConwayPolynomial<29, 11> { using ZPZ = aerobus::zpz<29>; using type =
  04364
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        ZPZV<8>, ZPZV<27»; }; // NOLINT
   template<> struct ConwayPolynomial<29, 12> { using ZPZ = aerobus::zpz<29>; using type =
04365
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<28>, ZPZV<29>, ZPZV<16>, 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<0
                                          ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<27»; }; // NOLINT</pre>
                                        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<14>, ZPZV<10>,
ZPZV<21>, ZPZV<18>, ZPZV<27>, ZPZV<5>, ZPZV<2»; }; // NOLINT</pre>
 04367
                                                                     template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::zpz<29>; using type
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        ZPZV<14>, ZPZV<8>, ZPZV<1>, ZPZV<12>, ZPZV<26>, ZPZV<27»; }; // NOLINT
    template<> struct ConwayPolynomial<29, 16> { using ZPZ = aerobus::zpz<29>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0</pre>
 04369
                                        ZPZV<2>, ZPZV<18>, ZPZV<23>, ZPZV<1>, ZPZV<27>, ZPZV<10>, ZPZV<20; }; // NOLINT
template<> struct ConwayPolynomial<29, 17> { using ZPZ = aerobus::zpz<29>; using type
                                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<27*; }; // NOLINT template<> struct ConwayPolynomial<29, 18> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<3, ZPZV<1>, ZPZV<3, ZPZV<1>, ZPZV<3, ZPZV<10>, ZPZV<10>, ZPZV<20>; ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<20>; Z
                                          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>
                                                                  template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
                                          POLYV<ZPZV<1>, ZPZV<28»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<31, 2> { using ZPZ = aerobus::zpz<31>; using type =
                          POLYV<ZPZV<1>, ZPZV<29>, ZPZV<3»; }; // NOLINT
                                          template<> struct ConwayPolynomial<31, 3> { using ZPZ = aerobus::zpz<31>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; // NOLINT
                         template<> struct ConwayPolynomial<31, 4> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<31, 5> { using ZPZ = aerobus::zpz<31>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; }; // NOLINT
04378
                                          template<> struct ConwayPolynomial<31, 6> { using ZPZ = aerobus::zpz<31>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<16>, ZPZV<8>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; // NOLINT
04379
                                             template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type
04380
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<12>, ZPZV<24>, ZPZV<3»; };
                           NOLINT
                                           template<> struct ConwayPolynomial<31, 9> { using ZPZ = aerobus::zpz<31>; using type =
04381
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<20>, ZPZV<29>, ZPZV<28»; };
                           // NOLINT
                                             template<> struct ConwayPolynomial<31, 10> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>, ZPZV<26>, ZPZV<13>, ZPZV<13>,
                           ZPZV<3»; }; // NOLINT</pre>
04383
                                             template<> struct ConwayPolynomial<31, 11> { using ZPZ = aerobus::zpz<31>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04384
                                             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 =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<28»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<31, 14> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYY<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<3»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04388
                          template<> struct ConwayPolynomial<31, 17> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                           template<> struct ConwayPolynomial<31, 18> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZ
04390
                           ZPZV<2>, ZPZV<7>, ZPZV<12>, ZPZV<11>, ZPZV<25>, ZPZV<20>, ZPZV<6>, ZPZV<3»; }; // NOLINT</pre>
04391
                                          template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; }; //</pre>
                           NOLINT
04392
                                             template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
                          POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                             template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
                          POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
04394
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
04395
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2»; };
                                                                                                                                                                                                                                                                                      // NOLINT
                                          template<> struct ConwayPolynomial<37, 5> { using ZPZ = aerobus::zpz<37>; using type =
04396
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35»; }; // NOLINT
04397
                                              template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<4>, ZPZV<30>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<37, 7> { using ZPZ = aerobus::zpz<37>; using type =
04398
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<35»; }; // NOLINT
                                              template<> struct ConwayPolynomial<37, 8> { using ZPZ = aerobus::zpz<37>; using type :
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<20>, ZPZV<27>, ZPZV<
04400
                                          template<> struct ConwayPolynomial<37, 9> { using ZPZ = aerobus::zpz<37>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<20>, ZPZV<32>, ZPZV<35»; };
                            // NOLINT
                                           template<> struct ConwayPolynomial<37, 10> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<11>, ZPZV<4>,
                            ZPZV<2»; }; // NOLINT</pre>
04402
                                          template<> struct ConwayPolynomial<37, 11> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<2>, ZPZV<35»; }; // NOLINT
  template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type
04403
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<31>, ZPZV<10>, ZPZV<23>, ZPZV<23>, ZPZV<18>, ZPZV<31>, ZPZV<33>, ZPZV<23>, ZPZV<33>, ZPZV<33
                                             template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<37, 14> { using ZPZ = aerobus::zpz<37>; using type
04405
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35>, ZPZV<35>, ZPZV<1>,
                          ZPZV<32>, ZPZV<16>, ZPZV<9>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<37, 15> { using ZPZ = aerobus::zpz<37>; using type
04406
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<31>,
                          ZPZV<28>, ZPZV<27>, ZPZV<13>, ZPZV<34>, ZPZV<33>, ZPZV<35»; }; // NOLINT
template<> struct ConwayPolynomial<37, 17> { using ZPZ = aerobus::zpz<37>; using type =
04407
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<35»; }; // NOLINT
template<> struct ConwayPolynomial<37, 18> { using ZPZ = aerobus::zpz<37>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<8>, ZPZV<15>,
                              ZPZV<1>, ZPZV<22>, ZPZV<20>, ZPZV<12>, ZPZV<32>, ZPZV<14>, ZPZV<27>, ZPZV<20>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<37, 19> { using ZPZ = aerobus::zpz<37>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<23>, ZPZV<35»; };</pre>
                                                  template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type =
                              POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
                              POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<15,; }; // NOLINT template<> struct ConwayPolynomial<41, 4> { using ZPZ = aerobus::zpz<41>; using type =
 04413
                            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
04414
                                                   template<> struct ConwayPolynomial<41, 6> { using ZPZ = aerobus::zpz<41>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>; }; // NOLINT template<> struct ConwayPolynomial<41, 7> { using ZPZ = aerobus::zpz<41>; using type =
 04416
                               \texttt{POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<35 **; }; \ // \texttt{NOLINT}  
                            template<> struct ConwayPolynomial<41, 8> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<32>, ZPZV<20>, ZPZV<6>, ZPZV<6>; };
04417
                                                   template<> struct ConwayPolynomial<41, 9> { using ZPZ = aerobus::zpz<41>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<5>, ZPZV<35»; };
                                // NOLINT
                              template<> struct ConwayPolynomial<41, 10> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<8>, ZPZV<80, ZPZV<30>,
04419
                               ZPZV<6»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<41, 11> { using ZPZ = aerobus::zpz<41>; using type
                               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<20>, ZPZV<35»; }; // NOLINT</pre>
                              \label{template} $$ \text{template}$ \sim \text{Struct ConwayPolynomial}$ < 41, 12> { using ZPZ = aerobus:: 2pz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<13>, ZPZV<34>, ZPZV<34>, ZPZV<24>, ZPZV<26>, ZPZV<13>, ZPZV<34>, ZPZV<34-, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<34, ZPZ
04421
                               ZPZV<21>, ZPZV<27>, ZPZV<6»; }; // NOLINT</pre>
                                                    template<> struct ConwayPolynomial<41, 13> { using ZPZ = aerobus::zpz<41>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<35»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<41, 14> { using ZPZ = aerobus::zpz<41>; using type =
04423
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<1>, ZPZV<1>, ZPZV<15>, ZPZV<15>, ZPZV<4>, ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35»; }; // NOLINT</pre>
04425
                                                 template<> struct ConwayPolynomial<41, 17> { using ZPZ = aerobus::zpz<41>; using type =
                              POLYY<ZPZV<0>, ZPZV<0>, ZPZV<0 , ZPZV<0 , ZPZV<0 , ZPZV<0
04426
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<7>, ZPZV<20>,
                               ZPZV<23>, ZPZV<35>, ZPZV<38>, ZPZV<24>, ZPZV<12>, ZPZV<29>, ZPZV<10>, ZPZV<6>, ZPZV<6»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZV<3 ,
04427
                               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 =
                              POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<43, 3> { using ZPZ = aerobus::zpz<43>; using type =
04430
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40»; }; // NOLINT template<> struct ConwayPolynomial<43, 4> { using ZPZ = aerobus::zpz<43>; using type =
 04431
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3>; // NOLINT
template<> struct ConwayPolynomial<43, 5> { using ZPZ = aerobus::zpz<43>; using type =
04432
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<40»; }; // NOLINT
 04433
                                                   template<> struct ConwayPolynomial<43, 6> { using ZPZ = aerobus::zpz<43>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<21>, ZPZV<21>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<43, 7> { using ZPZ = aerobus::zpz<43>; using type =
04434
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<7>, ZPZV<40»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<43, 8> { using ZPZ = aerobus::zpz<43>; using type =
 04435
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<39>, ZPZV<20>, ZPZV<24>, ZPZV<3»; }; //
                               NOLINT
                                                 template<> struct ConwayPolynomial<43, 9> { using ZPZ = aerobus::zpz<43>; using type =
04436
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<39>, ZPZV<10, ZPZV<40»; };
                                                 template<> struct ConwayPolynomial<43, 10> { using ZPZ = aerobus::zpz<43>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<26>, ZPZV<36>, ZPZV<5>, ZPZV<5>, ZPZV<27>, ZPZV<24>,
                               ZPZV<3»: }: // NOLINT
04438
                                                template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type =
                              POLYYCZPZV<1>, ZPZV<0>, ZPZV<0
                             template<> struct ConwayPolynomial<43, 12> { using ZPZ = aerobus::zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<27>, ZPZV<16>, ZPZV<17>, ZPZV<6>,
                               ZPZV<23>, ZPZV<38>, ZPZV<3»; }; // NOLINT</pre>
                              template<> struct ConwayPolynomial</a>13> { using ZPZ = aerobus::zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV
```

```
ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<40»; };</pre>
                                                                                                                                                                                                                                                            // NOLINT
                                template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<24>, ZPZV<37>, ZPZV<18>, ZPZV<4>, ZPZV<19>, ZPZV<3»; }; // NOLINT
                                template<> struct ConwayPolynomial<43, 15> { using ZPZ = aerobus::zpz<43>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>,
                                 ZPZV<22>, ZPZV<42>, ZPZV<4>, ZPZV<15>, ZPZV<37>, ZPZV<40»; }; // NOLINT</pre>
                                                        template<> struct ConwayPolynomial<43, 17> { using ZPZ = aerobus::zpz<43>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 template<> struct ConwayPolynomial<43, 18> { using ZPZ = aerobus::zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<24>, ZPZV<28>, ZPZV<41>, ZPZV<28>, ZPZV<41>, ZPZV<41>,
04444
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<24>, ZPZV<3>, ZPZV<3 , ZPZV<
                                                      template<> struct ConwayPolynomial<43, 19> { using ZPZ = aerobus::zpz<43>; using type
                                 POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  ZPZV<0>,
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40»; }; //</pre>
                                 NOLINT
04446
                                                      template<> struct ConwayPolynomial<47, 1> { using ZPZ = aerobus::zpz<47>; using type =
                                 POLYV<ZPZV<1>, ZPZV<42»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<47, 2> { using ZPZ = aerobus::zpz<47>; using type =
                                 POLYV<ZPZV<1>, ZPZV<45>, ZPZV<5»; }; // NOLINT
 04448
                                                      template<> struct ConwayPolynomial<47, 3> { using ZPZ = aerobus::zpz<47>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42»; }; // NOLINT

template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; }; // NOLINT
 04449
                                                       template<> struct ConwayPolynomial<47, 5> { using ZPZ = aerobus::zpz<47>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT
 04451
                                                      template<> struct ConwayPolynomial<47, 6> { using ZPZ = aerobus::zpz<47>; using type =
                                 \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<2>, \verb"ZPZV<35>, \verb"ZPZV<9>, \verb"ZPZV<41>, \verb"ZPZV<5>"; "// \verb"NOLINT" | NOLINT" | N
                               template<> struct ConwayPolynomial<47, 7> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT
04452
 04453
                                                       template<> struct ConwayPolynomial<47, 8> { using ZPZ = aerobus::zpz<47>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<29>, ZPZV<19>, ZPZV<3>, ZPZV<5»; };
                                                   template<> struct ConwayPolynomial<47, 9> { using ZPZ = aerobus::zpz<47>; using type
 04454
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<42»; };
                                  // NOLINT
                                template<> struct ConwayPolynomial<47, 10> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42>, ZPZV<14>, ZPZV<18>, ZPZV<45>, ZPZV<45>,
04455
                                 ZPZV<5»: }: // NOLINT
                                                    template<> struct ConwayPolynomial<47, 11> { using ZPZ = aerobus::zpz<47>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04457
04458
                                                       template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 \text{ZPZV}<0>, \text{ZPZV}<0>, \text{ZPZV}<5>, \text{ZPZV}<42»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<47, 14> { using ZPZ = aerobus::zpz<47>; using type =
04459
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV
                                 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
                                 ZPZV<31>, ZPZV<14>, ZPZV<42>, ZPZV<13>, ZPZV<17>, ZPZV<42»; }; // NOLINT
template<> struct ConwayPolynomial<47, 17> { using ZPZ = aerobus::zpz<47>; using type =
04461
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<42»; }; // NOLINT</pre>
                                                    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<41>, ZPZV<42>
                                 ZPZV<26>, ZPZV<44>, ZPZV<24>, ZPZV<22>, ZPZV<11>, ZPZV<5>, ZPZV<45>, ZPZV<33>, ZPZV<5»; }; // NOLINT
                                template<> struct ConwayPolynomial<47, 19> { using ZPZ = aerobus::zpz<47>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04463
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<42»; }; //</pre>
                                                      template<> struct ConwayPolynomial<53, 1> { using ZPZ = aerobus::zpz<53>; using type =
                                POLYV<ZPZV<1>, ZPZV<51»; }; // NOLINT
 04465
                                                      template<> struct ConwayPolynomial<53, 2> { using ZPZ = aerobus::zpz<53>; using type =
                                POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<53, 3> { using ZPZ = aerobus::zpz<53>; using type =
04466
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
 04467
                                template<> struct ConwayPolynomial<53, 5> { using ZPZ = aerobus::zpz<53>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT</pre>
 04468
                                                      template<> struct ConwayPolynomial<53, 6> { using ZPZ = aerobus::zpz<53>; using type =
04469
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<4>, ZPZV<45>, ZPZV<2»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<53, 7> { using ZPZ = aerobus::zpz<53>; using type
 04470
                                 \texttt{POLYV} < \texttt{ZPZV} < 1 >, \ \texttt{ZPZV} < 0 >, \ \texttt{ZPZV} <
04471
                                                    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 =
 04472
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<5>, ZPZV<51»; };
 04473
                                                    template<> struct ConwayPolynomial<53, 10> { using ZPZ = aerobus::zpz<53>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2», }; // NOLINT
 04474
                                                    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>
                                                 template<> struct ConwayPolynomial<53, 12> { using ZPZ = aerobus::zpz<53>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<34>, ZPZV<4>, ZPZV<13>, ZPZV<10>, ZPZV<42>, ZPZV<34>, ZPZV<34>, ZPZV<41>, ZPZV<40, ZPZV<42>, ZPZV<34>, ZPZV<34
                                                 template<> struct ConwayPolynomial<53, 13> { using ZPZ = aerobus::zpz<53>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              \text{ZPZV}<0>, \text{ZPZV}<52>, \text{ZPZV}<28>, \text{ZPZV}<51»; }; // NOLINT
                                               template<> struct ConwayPolynomial<53, 14> { using ZPZ = aerobus::zpz<53>; using type =
04477
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<5, ZPZV<45, ZPZV<45, ZPZV<23>, ZPZV<52>, ZPZV<0>, ZPZV<37>, ZPZV<12>, ZPZV<23>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<53, 15> { using ZPZ = aerobus::zpz<53>; using type =
04478
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>,
                               ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<53, 17> { using ZPZ = aerobus::zpz<53>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<51»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<53, 18> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<52>, ZPZV<51>, ZPZV<51>,
04480
                              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>, ZPZV<0>,
                              NOLINT
04482
                                                 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 =
04483
                             POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2»; }; // NOLINT
04484
                                                 template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; }; // NOLINT
template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
04485
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<2»; };
                                                                                                                                                                                                                                                                                                                  // NOLINT
                                                 template<> struct ConwayPolynomial<59, 5> { using ZPZ = aerobus::zpz<59>; using type =
04486
                             04487
                                                 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 :
04488
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<57»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<59, 8> { using ZPZ = aerobus::zpz<59>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<32>, ZPZV<2>, ZPZV<50>, ZPZV<2»; };
                            template<> struct ConwayPolynomial<59, 9> { using ZPZ = aerobus::zpz<59>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<32>, ZPZV<47>, ZPZV<57»; };</pre>
04490
                              // NOLINT
                             template<> struct ConwayPolynomial<59, 10> { using ZPZ = aerobus::zpz<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28>, ZPZV<25>, ZPZV<4>, ZPZV<39>, ZPZV<15>,
                              ZPZV<2»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<59, 11> { using ZPZ = aerobus::zpz<59>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<57»; }; // NOLINT</pre>
                                                  template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<21>, ZPZV<38>, ZPZV<8>, ZPZV<1>, ZPZV<2»; }; // NOLINT
04494
                                                 template<> struct ConwayPolynomial<59, 13> { using ZPZ = aerobus::zpz<59>; using type =
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                   template<> struct ConwayPolynomial<59, 14> { using ZPZ = aerobus::zpz<59>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<51>, ZPZV<11>,
                              ZPZV<13>, ZPZV<25>, ZPZV<32>, ZPZV<26>, ZPZV<2»; };  // NOLINT</pre>
04496
                                                  template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5 , ZPZV<5
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5</pre>
                            template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<27>, ZPZV<37>, ZPZV<38>, ZPZV<27>,
04498
                             ZPZV<11>, ZPZV<14>, ZPZV<7>, ZPZV<44>, ZPZV<16>, ZPZV<47>, ZPZV<34>, ZPZV<32>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<59, 19> { using ZPZ = aerobus::zpz<59>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>
04500
                                                 template<> struct ConwayPolynomial<61, 1> { using ZPZ = aerobus::zpz<61>; using type =
                             POLYV<ZPZV<1>, ZPZV<59»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<61, 2> { using ZPZ = aerobus::zpz<61>; using type =
04501
                              POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; }; // NOLINT
                                               template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
04502
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; }; // NOLINT template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
04503
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2»; }; // NOLINT

template<> struct ConwayPolynomial<61, 5> { using ZPZ = aerobus::zpz<61>; using type =
04504
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<61, 6> { using ZPZ = aerobus::zpz<61>; using type =
                                                                                                                                                                                                                                                                                                                                                                                                              // NOLINT
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<3>, ZPZV<29>, ZPZV<2»; };
                           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<5, ZP
04506
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<1>, ZPZV<1>, ZPZV<56>, ZPZV<2»; }; // NOLINT
                             template<> struct ConwayPolynomial<61, 9> { using ZPZ = aerobus::zpzc61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<50>,
                              // NOLINT
04509
                             template<> struct ConwayPolynomial<61, 10> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<15>, ZPZV<44>, ZPZV<16>, ZPZV<6>,
                              ZPZV<2»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<61, 11> { using ZPZ = aerobus::zpz<61>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            template<> struct ConwayPolynomial<61, 12> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<42>, ZPZV<33>, ZPZV<8>, ZPZV<38>, ZPZV<14>, ZPZV<15>, ZPZV<2»; }; // NOLINT
04511
                                                  template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<59»; };</pre>
                                                                                                                                                                                                                                         // NOLINT
                             template<> struct ConwayPolynomial<61, 14> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<48>, ZPZV<26>, ZPZV<11>,
                              ZPZV<8>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<2»; }; // NOLINT</pre>
                                                   template<> struct ConwayPolynomial<61, 15> { using ZPZ = aerobus::zpz<61>; using type
                            template<> struct ConwayPolynomial<61, 15> { using APA = defodus::2p2xo1>; using Lype - PolyV<2P2V<1>, ZP2V<0>, ZP2V<0>, ZP2V<0>, ZP2V<0>, ZP2V<0>, ZP2V<0>, ZP2V<3>, ZP2V<3>, ZP2V<3>, ZP2V<4>, ZP2V<4>, ZP2V<3>, ZP2V<3>, ZP2V<3>, ZP2V<44>, ZP2V<25>, ZP2V<23>, ZP2V<21>, ZP2V<51>, ZP2V<59»; }; // NoLINT template<> struct ConwayPolynomial<61, 17> { using ZPZ = aerobus::zp2<61>; using type = PolyV<ZPZV<1>, ZP2V<0>, ZP2V<0>, ZPZV<0>, ZPZV<0 , ZPZ
04515
                             ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZV<0>; ZPZ
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<35>, ZPZV<36>, ZPZV<13>,
                             ZPZV<36>, ZPZV<4>, ZPZV<32>, ZPZV<57>, ZPZV<42>, ZPZV<25>, ZPZV<25>, ZPZV<52>, ZPZV<22; }; // NOLINT
template<> struct ConwayPolynomial<61, 19> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; }; //</pre>
                              NOLINT
                                                  template<> struct ConwayPolynomial<67, 1> { using ZPZ = aerobus::zpz<67>; using type =
04518
                             POLYV<ZPZV<1>, ZPZV<65»; }; // NOLINT
04519
                                                  template<> struct ConwayPolynomial<67, 2> { using ZPZ = aerobus::zpz<67>; using type =
                             POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
04520
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65»; }; // NOLINT
04521
                                                  template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<54>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<67, 5> { using ZPZ = aerobus::zpz<67>; using type =
04522
                            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
04523
                                                   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<0>, ZPZV<7>, ZPZV<65»; };
04525
                                              template<> struct ConwayPolynomial<67, 8> { using ZPZ = aerobus::zpz<67>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<46>, ZPZV<17>, ZPZV<64>, ZPZV<64>, ZPZV<2»; }; //
                             NOLINT
                                                template<> struct ConwayPolynomial<67, 9> { using ZPZ = aerobus::zpz<67>; using type =
04526
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<49>, ZPZV<55>, ZPZV<65»; };
04527
                                                template<> struct ConwayPolynomial<67, 10> { using ZPZ = aerobus::zpz<67>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<0>, ZPZV<16>, ZPZV<7>, ZPZV<23>, ZPZV<2»; }; // NOLINT
04528
                                                  template<> struct ConwayPolynomial<67, 11> { using ZPZ = aerobus::zpz<67>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<9>, ZPZV<65»; };</pre>
                                                                                                                                                  // NOLINT
                             template<> struct ConwayPolynomial<67, 12> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<55>, ZPZV<64>,
                              ZPZV<21>, ZPZV<27>, ZPZV<2»; }; // NOLINT
                                                template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type =
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                               template<> struct ConwayPolynomial<67, 14> { using ZPZ = aerobus::zpz<67>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<17>, ZPZV<22>, ZPZV<5>,
                             ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type =
04532
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6 ), 
                                                  template<> struct ConwayPolynomial<67, 17> { using ZPZ = aerobus::zpz<67>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<65»; }; // NOLINT
template<> struct ConwayPolynomial<67, 18> { using ZPZ = aerobus::zpz<67>; using type
04534
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<5>, ZPZV<5-, ZPZV<5
                                              template<> struct ConwayPolynomial<67, 19> { using ZPZ = aerobus::zpz<67>; using type =
04535
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<18>, ZPZV<65»; }; //</pre>
                              NOLINT
04536
                                                  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 =
04538
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; }; // NOLINT template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
04539
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, ZPZV<7»; };
                                          template<> struct ConwayPolynomial<71, 5> { using ZPZ = aerobus::zpz<71>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<64»; }; // NOLINT
04541
                                         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 template<> struct ConwayPolynomial<71, 7> { using ZPZ = aerobus::zpz<71>; using type
04542
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<64»; }; //
                                          template<> struct ConwayPolynomial<71, 8> { using ZPZ = aerobus::zpz<71>; using type
04543
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<22>, ZPZV<19>, ZPZV<7»; };
                        template<> struct ConwayPolynomial<71, 9> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43>, ZPZV<43>, ZPZV<62>, ZPZV<64»; };
04544
                         // NOLINT
                                           template<> struct ConwayPolynomial<71, 10> { using ZPZ = aerobus::zpz<71>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<17>, ZPZV<26>, ZPZV<15, ZPZV<40>,
                          ZPZV<7»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<71, 11> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                           template<> struct ConwayPolynomial<71, 12> { using ZPZ = aerobus::zpz<71>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<24>, 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 =
04548
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<27>, ZPZV<64*; }; // NOLINT template<> struct ConwayPolynomial<71, 15> { using ZPZ = aerobus::zpz<71>; using type
                         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 , ZPZV<0
04550
                        template<> struct ConwayPolynomial<71, 19> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
04551
                          ZPZV<0>, 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 =
04552
                        POLYV<ZPZV<1>, ZPZV<68»; }; // NOLINT
                                          template<> struct ConwayPolynomial<73, 2> { using ZPZ = aerobus::zpz<73>; using type =
                        POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5»; }; // NOLINT
                                        template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68»; }; // NOLINT template<> struct ConwayPolynomial<73, 4> { using ZPZ = aerobus::zpz<73>; using type =
04555
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<73, 5> { using ZPZ = aerobus::zpz<73>; using type =
04556
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<68»; }; // NOLINT
04557
                                          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
04558
                                        template<> struct ConwayPolynomial<73, 7> { using ZPZ = aerobus::zpz<73>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<68»; }; // NOLINT
                                        template<> struct ConwayPolynomial<73, 8> { using ZPZ = aerobus::zpz<73>; using type =
04559
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<5>, ZPZV<39>, ZPZV<18>, ZPZV<5»; };
04560
                                       template<> struct ConwayPolynomial<73, 9> { using ZPZ = aerobus::zpz<73>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<72>, ZPZV<15>, ZPZV<68»; };
                                          template<> struct ConwayPolynomial<73, 10> { using ZPZ = aerobus::zpz<73>; using type =
04561
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<23>, ZPZV<33>, ZPZV<32>, ZPZV<69>,
                         ZPZV<5»: }: // NOLINT
                        template<> struct ConwayPolynomial<73, 11> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                         ZPZV<5>, ZPZV<68»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<73, 12> { using ZPZ = aerobus::zpz<73>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<52>, ZPZV<52>, ZPZV<26>, ZPZV<26>, ZPZV<46>, ZPZV<46>, ZPZV<29>, ZPZV<25>, ZPZV<5>; }; // NOLINT
                                       template<> struct ConwayPolynomial<73, 13> { using ZPZ = aerobus::zpz<73>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         \mbox{ZPZV}<0> , \mbox{ZPZV}<0> , \mbox{ZPZV}<7> , \mbox{ZPZV}<68 ); // NOLINT
                                        template<> struct ConwayPolynomial<73, 15> { using ZPZ = aerobus::zpz<73>; using type =
04565
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                                          template<> struct ConwayPolynomial<73, 17> { using ZPZ = aerobus::zpz<73>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<68»; }; // NOLINT
template<> struct ConwayPolynomial<73, 19> { using ZPZ = aerobus::zpz<73>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04567
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68»; }; //</pre>
                                          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 =
04569
                        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
04571
                                       template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<79, 5> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<76»; }; // NOLINT
```

```
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<4>, ZPZV<76»; }; // NOLINT
                                                template<> struct ConwayPolynomial<79, 8> { using ZPZ = aerobus::zpz<79>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<60>, ZPZV<59>, ZPZV<48>, ZPZV<3»; }; //
                             NOLINT
04576
                                                   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<5>, ZPZV<5-, ZPZV<5
                               // NOLINT
04577
                                                 template<> struct ConwayPolynomial<79, 10> { using ZPZ = aerobus::zpz<79>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<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
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<3>, ZPZV<76»; ); // NOLINT
template<> struct ConwayPolynomial<79, 12> { using ZPZ = aerobus::zpz<79>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<45>, ZPZV<55>, ZPZV<55>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<78>, ZPZV<4>, ZPZV<76»; };</pre>
                                                                                                                                                                                                                                            // NOLINT
                                                template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type =
04581
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZ
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      ZPZV<0>.
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<76»; }; //</pre>
                              NOLINT
04583
                                                 template<> struct ConwayPolynomial<83, 1> { using ZPZ = aerobus::zpz<83>; using type =
                             POLYV<ZPZV<1>, ZPZV<81»; }; // NOLINT
04584
                                                   template<> struct ConwayPolynomial<83, 2> { using ZPZ = aerobus::zpz<83>; using type =
                              POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; }; // NOLINT
04585
                                               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 =
04586
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2; }; // NOLINT template<> struct ConwayPolynomial<83, 5> { using ZPZ = aerobus::zpz<83>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<81»; }; // NOLINT
                                                template<> struct ConwayPolynomial<83, 6> { using ZPZ = aerobus::zpz<83>; using type =
04588
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<76>, ZPZV<32>, ZPZV<17>, ZPZV<2»; }; // NOLINT
                            template<> struct ConwayPolynomial<83, 7> { using ZPZ = aerobus::zpz<83>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT template<> struct ConwayPolynomial<83, 8> { using ZPZ = aerobus::zpz<83>; using type =
04589
04590
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<23>, ZPZV<42>, ZPZV<42»; };
                              NOLINT
04591
                                               template<> struct ConwayPolynomial<83, 9> { using ZPZ = aerobus::zpz<83>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<24>, ZPZV<18>, ZPZV<81»; };
                               // NOLINT
                                                 template<> struct ConwayPolynomial<83, 10> { using ZPZ = aerobus::zpz<83>; using type =
04592
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5 , ZPZV<5
                              ZPZV<2»; }; // NOLINT</pre>
04593
                                               template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04594
                                                  template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type =
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<65>, ZPZV<35>, ZPZV<31>, ZPZV<19>, ZPZV<65>, ZPZV<55>, ZPZV<75>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<81»; }; // NOLINT
    template<> struct ConwayPolynomial<83, 17> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                              template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              NOLINT
                                                 template<> struct ConwayPolynomial<89, 1> { using ZPZ = aerobus::zpz<89>; using type =
                             POLYV<ZPZV<1>, ZPZV<86»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<89, 2> { using ZPZ = aerobus::zpz<89>; using type =
                            POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; }; // NOLINT
04600
                                                  template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<86»; }; // NOLINT template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
04601
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<89, 5> { using ZPZ = aerobus::zpz<89>; using type =
04602
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<86»; }; // NOLINT
04603
                                                  template<> struct ConwayPolynomial<89, 6> { using ZPZ = aerobus::zpz<89>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<8>, ZPZV<8>, ZPZV<8>, ZPZV<15>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 
04604
                                                 template<> struct ConwayPolynomial<89, 8> { using ZPZ = aerobus::zpz<89>; using type
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<40>, ZPZV<79>, ZPZV<3»; };
                             template<> struct ConwayPolynomial<89, 9> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<12>, ZPZV<12>, ZPZV<86»; };</pre>
04606
```

```
// NOLINT
                     template<> struct ConwayPolynomial<89, 10> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<33>, ZPZV<82>, ZPZV<52>, ZPZV<4>,
04607
                     ZPZV<3»; }; // NOLINT</pre>
04608
                                    template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<88>, ZPZV<26>, ZPZV<86»; }; // NOLINT
                                    template<> struct ConwayPolynomial<89, 12> { using ZPZ = aerobus::zpz<89>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<85>, ZPZV<15>, ZPZV<44>, ZPZV<51>, ZPZV<8>, ZPZV<70>, ZPZV<52>, ZPZV<3»; }; // NOLINT
04610
                                  template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<86»; }; // NOLINT
template<> struct ConwayPolynomial<89, 17> { using ZPZ = aerobus::zpz<89>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      template<> struct ConwayPolynomial<89, 19> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04613
                                    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 =
04614
                     POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5»; }; // NOLINT
04615
                                    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 =
04616
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; }; // NOLINT
                    template<> struct ConwayPolynomiale97, 5> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<92»; }; // NOLINT
template<> struct ConwayPolynomiale97, 6> { using ZPZ = aerobus::zpz<97>; using type =
04617
04618
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<92>, ZPZV<58>, ZPZV<88>, ZPZV<58; }; // NOLINT
                                    template<> struct ConwayPolynomial<97, 7> { using ZPZ = aerobus::zpz<97>; using type =
04619
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<92»; };
04620
                                    template<> struct ConwayPolynomial<97, 8> { using ZPZ = aerobus::zpz<97>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<1>, ZPZV<32>, ZPZV<5»; };
                                   template<> struct ConwayPolynomial<97, 9> { using ZPZ = aerobus::zpz<97>; using type
04621
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59>, ZPZV<7>, ZPZV<92»; };
                                   template<> struct ConwayPolynomial<97, 10> { using ZPZ = aerobus::zpz<97>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<20>,
                     ZPZV<5»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04623
                                  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
04625
                                  template<> struct ConwayPolynomial<97, 13> { using ZPZ = aerobus::zpz<97>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<92»; }; // NOLINT
template<> struct ConwayPolynomial<97, 19> { using ZPZ = aerobus::zpz<97>; using type
04627
                     POLYYCZPZV<1>, ZPZV<0>, ZPZV<0
                                    template<> struct ConwayPolynomial<101, 1> { using ZPZ = aerobus::zpz<101>; using type =
                     POLYV<ZPZV<1>, ZPZV<99»; }; // NOLINT
                                  template<> struct ConwayPolynomial<101, 2> { using ZPZ = aerobus::zpz<101>; using type =
04629
                    POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; }; // NOLINT
04630
                                    template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
                     POLYY<ZPZY<1>, ZPZV<0>, ZPZV<3>, ZPZV<9»; }; // NOLINT template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
04631
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<78>, ZPZV<2»; }; // NOLINT
04632
                                    template<> struct ConwayPolynomial<101, 5> { using ZPZ = aerobus::zpz<101>; using type =
                     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 =
04633
                    POLYV<2PZV<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 =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<9»; }; // NOLINT
04635
                                  template<> struct ConwayPolynomial<101, 8> { using ZPZ = aerobus::zpz<101>; using type =
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76>, ZPZV<29>, ZPZV<24>, ZPZV<24>, ZPZV<2, ZPZV<29>, ZPZV<24>, ZPZV<20>; }; //
                     NOLINT
                                    template<> struct ConwayPolynomial<101, 9> { using ZPZ = aerobus::zpz<101>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64>, ZPZV<47>, ZPZV<99»; };
                                  template<> struct ConwayPolynomial<101, 10> { using ZPZ = aerobus::zpz<101>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<6>, ZPZV<4>, ZPZV<40>, ZPZV<100>, ZPZV<100>, ZPZV<52>,
                     ZPZV<2»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<101, 11> { using ZPZ = aerobus::zpz<101>; using type :
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<31>, ZPZV<99»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<101, 12> { using ZPZ = aerobus::zpz<101>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<79>, ZPZV<64>, ZPZV<39>, ZPZV<38>, ZPZV<48>,
                      ZPZV<84>, ZPZV<21>, ZPZV<2»; }; // NOLINT</pre>
```

```
template<> struct ConwayPolynomial<101, 13> { using ZPZ = aerobus::zpz<101>; using type :
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<99»; };</pre>
                                                                                                                                                                                                       // NOLINT
                                       template<> struct ConwayPolynomial<101, 17> { 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<31>, ZPZV<99»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                     // NOLINT
                                          template<> struct ConwayPolynomial<101, 19> { using ZPZ = aerobus::zpz<101>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<99»; }; //</pre>
                         NOLINT
04643
                                         template<> struct ConwayPolynomial<103, 1> { using ZPZ = aerobus::zpz<103>; using type =
                         POLYV<ZPZV<1>, ZPZV<98»; }; // NOLINT
                                          template<> struct ConwayPolynomial<103, 2> { using ZPZ = aerobus::zpz<103>; using type =
04644
                         POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; }; // NOLINT
04645
                                         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 =
04646
                        POLYVCZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<88>, ZPZV<5>; }; // NOLINT template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<9*, ZPZV<11, ZPZV<98; }; // NOLINT
                                           template<> struct ConwayPolynomial<103, 6> { using ZPZ = aerobus::zpz<103>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<9b, ZPZV<30>, ZPZV<5»; }; // NOLINT
                                        template<> struct ConwayPolynomial<103, 7> { using ZPZ = aerobus::zpz<103>; using type =
04649
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98»; }; // NOLINT
04650
                                         template>> struct ConwayPolynomial<103, 8> { using ZPZ = aerobus::zpz<103>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<70>, ZPZV<71>, ZPZV<49>, ZPZV<49>, ZPZV<5»; }; //
04651
                                        template<> struct ConwayPolynomial<103, 9> { using ZPZ = aerobus::zpz<103>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<98»; };
                          // NOLINT
                                         template<> struct ConwayPolynomial<103, 10> { using ZPZ = aerobus::zpz<103>; using type =
04652
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1), ZPZV<101>, ZPZV<86>, ZPZV<101>, ZPZV<94>, ZPZV<11>,
                          ZPZV<5»; }; // NOLINT
                                       template<> struct ConwayPolynomial<103, 11> { using ZPZ = aerobus::zpz<103>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<5>, ZPZV<98»; }; // NOLINT
  template<> struct ConwayPolynomial<103, 12> { using ZPZ = aerobus::zpz<103>; using type :
04654
                         POLYVCZPZVC1>, ZPZVC0>, ZPZVC0>, ZPZVC1>, ZPZVC74>, ZPZVC23>, ZPZVC94>, ZPZVC20>, ZPZVC81>, ZPZVC29>, ZPZVC88>, ZPZVC88>, ZPZVC88>, ZPZVC88>, ZPZVC5»; }; // NOLINT
                                         template<> struct ConwayPolynomial<103, 13> { using ZPZ = aerobus::zpz<103>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          template<> struct ConwayPolynomial<103, 17> { using ZPZ = aerobus::zpz<103>; using type
04656
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         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<2>, ZPZV<2>, ZPZV<98»; }; //</pre>
                         NOLINT
04658
                                          template<> struct ConwavPolynomial<107. 1> { using ZPZ = aerobus::zpz<107>; using type =
                         POLYV<ZPZV<1>, ZPZV<105»; }; // NOLINT
                                           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 =
04660
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<105»; }; // NOLINT template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
04661
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; };
                                                                                                                                                                                                                                                                         // NOLINT
                                         template<> struct ConwayPolynomial<107, 5> { using ZPZ = aerobus::zpz<107>; using type =
04662
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<105»; }; // NOLINT
04663
                                           template<> struct ConwayPolynomial<107, 6> { using ZPZ = aerobus::zpz<107>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<52>, ZPZV<22>, ZPZV<79>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type =
04664
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<105»; }; // NOLINT
                                          template<> struct ConwayPolynomial<107, 8> { using ZPZ = aerobus::zpz<107>; using type
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<24>, ZPZV<95>, ZPZV<2»; };
                         NOLINT
                                          template<> struct ConwayPolynomial<107, 9> { using ZPZ = aerobus::zpz<107>; using type
04666
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<36>, ZPZV<105»; };
                                            template<> struct ConwayPolynomial<107, 10> { using ZPZ = aerobus::zpz<107>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<94>, ZPZV<61>, ZPZV<83>, ZPZV<83>, ZPZV<85>,
                         ZPZV<2»; }; // NOLINT</pre>
                         // NOLINT
                         ZPZV<8>, ZPZV<105»; };</pre>
                                           template<> struct ConwayPolynomial<107, 12> { using ZPZ = aerobus::zpz<107>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<61>,
                          ZPZV<42>, ZPZV<57>, ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105»; };</pre>
                                                                                                                                                                                                           // NOLINT
                                           template<> struct ConwayPolynomial<107, 17> { using ZPZ = aerobus::zpz<107>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                    // NOLINT
                         template<> struct ConwayPolynomial<107, 19> { using ZPZ = aerobus::zpz<107>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04672
```

```
NOLINT
                                       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 =
 04674
                       POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; }; // NOLINT
                                       template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
04675
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT
                                        template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
 04676
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<109, 5> { using ZPZ = aerobus::zpz<109>; using type =
 04677
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103»; }; // NOLINT
                      template<> struct ConwayPolynomial<109, 6> { using ZPZ = aerobus::zpz<109>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<102>, ZPZV<66>, ZPZV<6»; }; // NOLINT
 04678
                                         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<14>, ZPZV<143, ZPZV<103»; };
 04680
                                      template<> struct ConwayPolynomial<109, 8> { using ZPZ = aerobus::zpz<109>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<34>, ZPZV<86>, ZPZV<6»; }; //
                        NOLINT
                                       template<> struct ConwayPolynomial<109, 9> { using ZPZ = aerobus::zpz<109>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<93>, ZPZV<87>, ZPZV<103»; };
04682
                                      template<> struct ConwayPolynomial<109, 10> { using ZPZ = aerobus::zpz<109>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<71>, ZPZV<55>, ZPZV<16>, ZPZV<16>, ZPZV<75>, ZPZV<69>,
                        ZPZV<6»; }; // NOLINT</pre>
04683
                                       template<> struct ConwayPolynomial<109, 11> { using ZPZ = aerobus::zpz<109>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<11>, ZPZV<103»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<109, 12> { using ZPZ = aerobus::zpz<109>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<50>, ZPZV<53>, ZPZV<37>, ZPZV<65>, ZPZV<65>,
                        ZPZV<103>, ZPZV<28>, ZPZV<6»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<109, 13> { using ZPZ = aerobus::zpz<109>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<103»; };</pre>
                                                                                                                                                                                            // NOLINT
                                     template<> struct ConwayPolynomial<109, 17> { using ZPZ = aerobus::zpz<109>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04687
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<15>, ZPZV<103»; }; //</pre>
                                      template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
                        POLYV<ZPZV<1>, ZPZV<110»; }; // NOLINT
                                       template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
04689
                       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 =
 04691
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<62>, ZPZV<62>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<113, 5> { using ZPZ = aerobus::zpz<113>; using type =
04692
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110»; }; // NOLINT
04693
                                        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
 04694
                                      template<> struct ConwayPolynomial<113, 7> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<510»; }; // NOLINT template<> struct ConwayPolynomial<113, 8> { using ZPZ = aerobus::zpz<113>; using type =
04695
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<38>, ZPZV<38>, ZPZV<28>, ZPZV<28, ZPZV<3»; };
                                     template<> struct ConwayPolynomial<113, 9> { using ZPZ = aerobus::zpz<113>; using type =
 04696
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<8110»; };
                        // NOLINT
04697
                                       template<> struct ConwayPolynomial<113, 10> { using ZPZ = aerobus::zpz<113>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<56>,
                        ZPZV<3»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<3>, ZPZV<110»; }; // NOLINT</pre>
                        template<> struct ConwayPolynomial<113, 12> { using ZPZ = aerobus::zpz<113>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<23>, ZPZV<62>, ZPZV<4>, ZPZV<98>, ZPZV<56>,
ZPZV<10>, ZPZV<27>, ZPZV<3»; }; // NOLINT</pre>
04699
                                         template<> struct ConwayPolynomial<113,
                                                                                                                                                                                                    13> { using ZPZ = aerobus::zpz<113>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        template<> struct ConwayPolynomial<113, 17> { using ZPZ = aerobus::zpz<113>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04701
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4110s; }; // NOLINT
template<> struct ConwayPolynomial<113, 19> { using ZPZ = aerobus::zpz<113>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<2>, ZPZV<22</pre>
                        NOLINT
04703
                                      template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
                       POLYV<ZPZV<1>, ZPZV<124»; }; // NOLINT
                                        template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
                       POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3»; }; // NOLINT
 04705
                                     template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124»; }; // NOLINT template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<97>, ZPZV<3»; }; // NOLINT
 04706
```

```
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
                                         template<> struct ConwayPolynomial<127, 6> { using ZPZ = aerobus::zpz<127>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, 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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<124»; }; // NOLINT
                                          template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<104>, ZPZV<55>, ZPZV<8>, ZPZV<3»; };
04711
                                         template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<119>, ZPZV<126>, ZPZV<124»;
                           }; // NOLINT
04712
                                             template<> struct ConwayPolynomial<127, 10> { using ZPZ = aerobus::zpz<127>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<107>, ZPZV<64>, ZPZV<95>, ZPZV<60>, ZPZV<4>,
                           ZPZV<3»; }; // NOLINT</pre>
04713
                                          template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<11>, ZPZV<124»; }; // NOLINT 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>
04715
                                            template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<124»; }; // NOLINT template<> struct ConwayPolynomial<127, 17> { using ZPZ = aerobus::zpz<127>; using type =
04716
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           template<> struct ConwayPolynomial<127, 19> { using ZPZ = aerobus::zpz<127>; using type =
04717
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3, ZPZV<0>, ZPZV<3, ZP
                           NOLINT
04718
                                             template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
                          POLYV<ZPZV<1>, ZPZV<129»; }; // NOLINT
04719
                                         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 =
 04720
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<129»; ); // NOLINT template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<131, 5> { using ZPZ = aerobus::zpz<131>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<129»; }; // NOLINT
04723
                                           POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<66>, ZPZV<66>, ZPZV<42>, ZPZV<2>, ZPZV<2>; ZPZV<2>, ZPZV<2>; J; // NOLINT template<> struct ConwayPolynomial<131, 7> { using ZPZ = aerobus::zpz<131>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<10>, ZPZV<10>,
 04725
                                          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
04726
                                         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
                                            \texttt{template<> struct ConwayPolynomial<131, 10> \{ using ZPZ = aerobus::zpz<131>; using type } \\
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<9>, ZPZV<9>, ZPZV<426>, ZPZV<44>,
                           ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         template<> struct ConwayPolynomial<131, 12> { using ZPZ = aerobus::zpz<131>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<12>, ZPZV<40>, ZPZV<40, ZPZV<125>,
                           ZPZV<28>, ZPZV<103>, ZPZV<2»; }; // NOLINT</pre>
04730
                                          template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<129»; };</pre>
                                                                                                                                                                                                              // NOLINT
                                            template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0
                          template<> struct ConwayPolynomial<131, 19> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
                           ZPZV<0>, ZPZV<129»; }; //</pre>
                           NOLINT
                                             template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
                          POLYV<ZPZV<1>, ZPZV<134»; }; // NOLINT
                                           template<> struct ConwayPolynomial<137, 2> { using ZPZ = aerobus::zpz<137>; using type =
04734
                          POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type =
04735
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134»; }; // NOLINT
                                         template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
 04736
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<95>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
 04737
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 6> { using ZPZ = aerobus::zpz<137>; using type = POLYV<2PZV<1>, ZPZV<1>, ZPZV<116>, ZPZV<102>, ZPZV<3>, ZPZV<3»; }; // NOLINT
 04738
                                           template<> struct ConwayPolynomial<137, 7> { using ZPZ = aerobus::2pz<137>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<134»; }; // NOLINT
04740
                                         template<> struct ConwayPolynomial<137, 8> { using ZPZ = aerobus::zpz<137>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105>, ZPZV<21>, ZPZV<34>, ZPZV<33»; }; //
```

```
template<> struct ConwayPolynomial<137, 9> { using ZPZ = aerobus::zpz<137>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<80>, ZPZV<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<2>, ZPZV<20>, ZPZV<67>, ZPZV<93>, ZPZV<119>,
04742
                           ZPZV<3»: }: // NOLINT
                                             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>
                           \label{eq:convayPolynomial<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<36>, ZPZV<36>, ZPZV<50>, ZPZV<10>, Z
                           ZPZV<135>, ZPZV<61>, ZPZV<3»: }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<137, 13> { using ZPZ = aerobus::zpz<137>; using type
04745
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<134»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<137, 17> { using ZPZ = aerobus::zpz<137>; using type =
04746
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<134»; }; //</pre>
                           NOLINT
04748
                                            template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
                           POLYV<ZPZV<1>, ZPZV<137»; }; // NOLINT
04749
                                             template<> struct ConwayPolynomial<139, 2> { using ZPZ = aerobus::zpz<139>; using type =
                           POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<139, 3> { using ZPZ = aerobus::zpz<139>; using type =
04750
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<137»; }; // NOLINT
                                             template<> struct ConwayPolynomial<139, 4> { using ZPZ = aerobus::zpz<139>; using type =
 04751
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2»; }; // NOLINT

template<> struct ConwayPolynomial<139, 5> { using ZPZ = aerobus::zpz<139>; using type =
04752
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<137»; }; // NOLINT
                                             template<> struct ConwayPolynomial<139, 6> { using ZPZ = aerobus::zpz<139>; using type =
 04753
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<46>, ZPZV<10>, ZPZV<118>, ZPZV<2»; }; // NOLINT
 04754
                                             template<> struct ConwayPolynomial<139, 7> { using ZPZ = aerobus::zpz<139>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<137»; }; // NOLINT
                                            template<> struct ConwayPolynomial<139, 8> { using ZPZ = aerobus::zpz<139>; using type =
04755
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>, ZPZV<36>, ZPZV<21>, ZPZV<2»; };
                           NOLINT
                                           template<> struct ConwayPolynomial<139, 9> { using ZPZ = aerobus::zpz<139>; using type =
04756
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<70>, ZPZV<87>, ZPZV<137»; };
                            // NOLINT
                                           template<> struct ConwayPolynomial<139, 10> { using ZPZ = aerobus::zpz<139>; using type =
04757
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<100>, ZPZV<100>, ZPZV<66>, ZPZV<106>, ZPZV<2»; }; // NOLINT
04758
                                             template<> struct ConwayPolynomial<139, 11> { using ZPZ = aerobus::zpz<139>; using type :
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<7>, ZPZV<137»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<139, 12> { using ZPZ = aerobus::zpz<139>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<120>, ZPZV<75>, ZPZV<41>, ZPZV<77>, ZPZV<70>, ZPZV<106>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
04759
                                            template<> struct ConwayPolynomial<139, 13> { using ZPZ = aerobus::zpz<139>; using type
                            \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ 
                           ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<137»; }; // NOLINT
template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type =
04761
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137»; };</pre>
                                           template<> struct ConwayPolynomial<139, 19> { using ZPZ = aerobus::zpz<139>; using type
 04762
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<137»; }; //</pre>
                           NOLINT
                                            template<> struct ConwayPolynomial<149, 1> { using ZPZ = aerobus::zpz<149>; using type =
04763
                           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
 04765
                                           template<> struct ConwayPolynomial<149, 3> { using ZPZ = aerobus::zpz<149>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
04766
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<2»; }; // NOLINT
                                              template<> struct ConwayPolynomial<149, 5> { using ZPZ = aerobus::zpz<149>; using type =
 04767
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<147»; }; // NOLINT
 04768
                                          template<> struct ConwayPolynomial<149, 6> { using ZPZ = aerobus::zpz<149>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<105>, ZPZV<33>, ZPZV<55>, ZPZV<2½; }; // NOLINT template<> struct ConwayPolynomial<149, 7> { using ZPZ = aerobus::zpz<149>; using type =
04769
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, 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<123>, ZPZV<2); };
                                           template<> struct ConwayPolynomial<149, 9> { using ZPZ = aerobus::zpz<149>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<146>, ZPZV<146>, ZPZV<20>, ZPZV<147»;
                           }; // NOLINT
                                               template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<74>, ZPZV<42>, ZPZV<148>, ZPZV<143>, ZPZV<51>,
                           ZPZV<2»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<149, 11> { using ZPZ = aerobus::zpz<149>; using type =
                            \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                            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<91>, ZPZV<52>, ZPZV<9>,
                             ZPZV<104>, ZPZV<110>, ZPZV<2»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<149, 13> { using ZPZ = aerobus::zpz<149>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             04777
                                                template<> struct ConwayPolynomial<149, 19> { using ZPZ = aerobus::zpz<149>; using type =
                            POLYV<ZPZV<0>, ZPZV<0>, ZPZV<0
                            NOLINT
                                                 template<> struct ConwayPolynomial<151, 1> { using ZPZ = aerobus::zpz<151>; using type =
                           POLYV<ZPZV<1>, ZPZV<145»; }; // NOLINT
04779
                                               template<> struct ConwayPolynomial<151, 2> { using ZPZ = aerobus::zpz<151>; using type =
                           POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6»; }; // NOLINT
                                                template<> struct ConwayPolynomial<151, 3> { using ZPZ = aerobus::zpz<151>; using type =
04780
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<145»; }; // NOLINT
                                                template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
04781
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6»; }; // NOLINT
04782
                                                template<> struct ConwayPolynomial<151, 5> { using ZPZ = aerobus::zpz<151>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, 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<12>, ZPZV<18>, ZPZV<15>, ZPZV<6»; }; // NOLINT
04783
                                                template<> struct ConwayPolynomial<151, 7> { using ZPZ = aerobus::2pz<151>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>; ;;
04785
                                              template<> struct ConwayPolynomial<151, 8> { using ZPZ = aerobus::zpz<151>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<140>, ZPZV<122>, ZPZV<43>, ZPZV<6»; }; //
                            NOLINT
                                               template<> struct ConwayPolynomial<151, 9> { using ZPZ = aerobus::zpz<151>; using type =
04786
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<126>, ZPZV<126>, ZPZV<145»;
                                                template<> struct ConwayPolynomial<151, 10> { using ZPZ = aerobus::zpz<151>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<21>, ZPZV<104>, ZPZV<49>, ZPZV<20>, ZPZV<142>,
                             ZPZV<6»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<151, 11> { using ZPZ = aerobus::zpz<151>; using type =
04788
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<1>, ZPZV<145»; };</pre>
                                                                                                                                                  // NOLINT
                                               template<> struct ConwayPolynomial<151, 12> { using ZPZ = aerobus::zpz<151>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<121>, ZPZV<101>, ZPZV<101>, ZPZV<6>, ZPZV<77>,
                            ZPZV<107>, ZPZV<147>, ZPZV<6»; }; // NOLINT
   template<> struct ConwayPolynomial<151, 13> { using ZPZ = aerobus::zpz<151>; using type :
04790
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<145»; }; // NOLINT</pre>
04791
                                                template<> struct ConwayPolynomial<151,
                                                                                                                                                                                                                                                  17> { using ZPZ = aerobus::zpz<151>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<145»; }; // NOLINT
template<> struct ConwayPolynomial<151, 19> { using ZPZ = aerobus::zpz<151>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
04792
                              ZPZV<0>, ZPZV<0>
04793
                                               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 =
04794
                            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 =
04796
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<136>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<157, 5> { using ZPZ = aerobus::zpz<157>; using type =
04797
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<152»; }; // NOLINT
04798
                                                 template<> struct ConwayPolynomial<157, 6> { using ZPZ = aerobus::zpz<157>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<130>, ZPZV<43>, ZPZV<144>, ZPZV<5»; }; // NOLINT
04799
                                             template<> struct ConwayPolynomial<157, 7> { using ZPZ = aerobus::zpz<157>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12*; }; // NOLINT template<> struct ConwayPolynomial<157, 8> { using ZPZ = aerobus::zpz<157>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<40>, ZPZV<153>, ZPZV<5*; }; //
04800
                            NOLINT
                                                 template<> struct ConwayPolynomial<157, 9> { using ZPZ = aerobus::zpz<157>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<114>, ZPZV<52>, ZPZV<152»;
                             }; // NOLINT
                            template<> struct ConwayPolynomial<157, 10> { using ZPZ = aerobus::zpz<157>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<61>, ZPZV<22>, ZPZV<124>, ZPZV<61>, ZPZV<93>,
04802
                             ZPZV<5»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<157, 11> { using ZPZ = aerobus::zpz<157>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<29>, ZPZV<152»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<157, 12> { using ZPZ = aerobus::zpz<157>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<77>, ZPZV<110>, ZPZV<72>, ZPZV<43>,
                             ZPZV<152>, ZPZV<57>, ZPZV<5»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<157, 13> { using ZPZ = aerobus::zpz<157>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             \mbox{ZPZV}\mbox{<0>, ZPZV}\mbox{<156>, ZPZV}\mbox{<9>, ZPZV}\mbox{<152}\mbox{»; }; // \mbox{NOLINT}
                            template<> struct ConwayPolynomial<157, 17> { using ZPZ = aerobus::zpz<157>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04806
```

```
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<14>, ZPZV<152»; }; //</pre>
                           NOLINT
04808
                                             template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
                          POLYV<ZPZV<1>, ZPZV<161»; }; // NOLINT
                                              template<> struct ConwayPolynomial<163, 2> { using ZPZ = aerobus::zpz<163>; using type =
                           POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; }; // NOLINT
04810
                                            template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<161»; }; // NOLINT template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
04811
                          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2x; }; // NOLINT template<> struct ConwayPolynomial<163, 5> { using ZPZ = aerobus::zpz<163>; using type =
04812
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<161»; }; // NOLINT
04813
                                           template<> struct ConwayPolynomial<163, 6> { using ZPZ = aerobus::zpz<163>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<25>, ZPZV<156>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<163, 7> { using ZPZ = aerobus::zpz<163>; using type =
04814
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<161»; }; // NOLINT
                                             template<> struct ConwayPolynomial<163, 8> { using ZPZ = aerobus::zpz<163>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<132>, ZPZV<83>, ZPZV<6>, ZPZV<2»; };
04816
                                            template<> struct ConwayPolynomial<163, 9> { using ZPZ = aerobus::zpz<163>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<161>, ZPZV<162>, ZPZV<162>, ZPZV<1261»;
                           }; // NOLINT
04817
                                              template<> struct ConwayPolynomial<163, 10> { using ZPZ = aerobus::zpz<163>; using type =
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3111>, ZPZV<120>, ZPZV<125>, ZPZV<15>, ZPZV<0>,
                           ZPZV<2»; }; // NOLINT</pre>
04818
                                            template<> struct ConwayPolynomial<163, 11> { using ZPZ = aerobus::zpz<163>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                           ZPZV<11>, ZPZV<161»; }; // NOLINT
                                            template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type =
04819
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<31>, ZPZV<38>, ZPZV<103>, ZPZV<10>, ZPZV<40>, ZPZV<30>, ZPZV<31>, ZPZV<31
, ZPZV
                                          template<> struct ConwayPolynomial<163, 13> { using ZPZ = aerobus::zpz<163>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<161»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<163, 17> { using ZPZ = aerobus::zpz<163>; using type =
04821
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<71>, ZPZV<711>, ZPZV<161»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<163, 19> { using ZPZ = aerobus::zpz<163>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8</pre>,  //
                           NOLINT
04823
                                             template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<162»; }; // NOLINT
04824
                                             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 =
04825
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162»; }; // NOLINT
                                             template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type =
04826
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; }; // NOLINT
                                              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
04828
                                            template<> struct ConwayPolynomial<167, 6> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<75>, ZPZV<38>, ZPZV<2>, ZPZV<5»; }; // NOLINT
                                             template<> struct ConwayPolynomial<167, 7> { using ZPZ = aerobus::zpz<167>; using type
04829
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162»; }; // NOLINT
                                           template<> struct ConwayPolynomial<167, 8> { using ZPZ = aerobus::zpz<167>; using type =
04830
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<149>, ZPZV<56>, ZPZV<113>, ZPZV<5»; };
                           NOLINT
04831
                                            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<0>, ZPZV<16>, ZPZV<16, Z
                           }; // NOLINT
                                               template<> struct ConwayPolynomial<167, 10> { using ZPZ = aerobus::zpz<167>; using type :
                           POLYV<ZPZV<1>, ZPZV<0>, ZPŽV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68>, ZPZV<68>, ZPZV<109>, ZPZV<143>,
                           ZPZV<148>, ZPZV<5»; }; // NOLINT</pre>
04833
                                             template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type :
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                              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<
                           04835
                                            template<> struct ConwayPolynomial<167, 13> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<167, 17> { using ZPZ = aerobus::zpz<167>; using type
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<32>, ZPZV<162»; }; // NOLINT
template<> struct ConwayPolynomial<167, 19> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                             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 =
04839
                        POLYV<2PZV<1>, ZPZV<169>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171»; }; // NOLINT
                    template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<2>; }; // NOLINT
                                 template<> struct ConwayPolynomial<173, 5> { using ZPZ = aerobus::zpz<173>; using type =
 04842
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171»; }; // NOLINT
04843
                                   template<> struct ConwayPolynomial<173, 6> { using ZPZ = aerobus::zpz<173>; using type =
                    POLYV<2PZV<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 =
 04844
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 8> { using ZPZ = aerobus::zpz<173>; using type =
 04845
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<125>, ZPZV<158>, ZPZV<27>, ZPZV<2»; }; //
                     NOLINT
                                  template<> struct ConwayPolynomial<173, 9> { using ZPZ = aerobus::zpz<173>; using type
04846
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<104>, ZPZV<171»;
                     }; // NOLINT
04847
                                   template<> struct ConwayPolynomial<173, 10> { using ZPZ = aerobus::zpz<173>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<164>, ZPZV<164>, ZPZV<48>, ZPZV<106>, ZPZV<58>, ZPZV<2»; }; // NOLINT
                                   template<> struct ConwayPolynomial<173, 11> { using ZPZ = aerobus::zpz<173>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<12>, ZPZV<171»; }; // NOLINT</pre>
04849
                                   template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type :
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<24>, ZPZV<46>, ZPZV<46>, ZPZV<166>, ZPZV<0>, ZPZV<159>, ZPZV<22>, ZPZV<22»; }; // NOLINT
04850
                                   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 =
04851
                    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<7>, ZPZV<7>, ZPZV<7>, ZPZV<7171»; }; // NOLINT
template<> struct ConwayPolynomial<173, 19> { using ZPZ = aerobus::zpz<173>; using type =
04852
                     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<6>, ZPZV<6</pre>, ZPZV<171»; }; //</pre>
04853
                                   template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
                    POLYV<ZPZV<1>, ZPZV<177»; }; // NOLINT
                                 template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
04854
                    POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2»; }; // NOLINT
                                   template<> struct ConwayPolynomial<179, 3> { using ZPZ = aerobus::zpz<179>; using type =
                    POLYY<ZPZY<1>, ZPZY<0>, ZPZY<4>, ZPZY<177%; }; // NOLINT template<> struct ConwayPolynomial<179, 4> { using ZPZ = aerobus::zpz<179>; using type =
 04856
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<179, 5> { using ZPZ = aerobus::zpz<179>; using type =
 04857
                    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»; };
 04859
                                template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<177»; }; // NOLINT
 04860
                                 template<> struct ConwayPolynomial<179, 8> { using ZPZ = aerobus::zpz<179>; using type =
                    POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<163>, ZPZV<144>, ZPZV<73>, ZPZV<2»; }; //
                                   template<> struct ConwayPolynomial<179, 9> { using ZPZ = aerobus::zpz<179>; using type =
 04861
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<40>, ZPZV<46>, ZPZV<177»; };
                      // NOLINT
04862
                                 template<> struct ConwayPolynomial<179, 10> { using ZPZ = aerobus::zpz<179>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<115>, ZPZV<71>, ZPZV<150>, ZPZV<49>, ZPZV<87>,
                      ZPZV<2»; }; // NOLINT</pre>
 04863
                                template<> struct ConwayPolynomial<179, 11> { using ZPZ = aerobus::zpz<179>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<28>, ZPZV<177»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<179, 12> { using ZPZ = aerobus::zpz<179>; using type =
04864
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<103>, ZPZV<83>, ZPZV<43>, ZPZV<76>, ZPZV<8>, ZPZV<177>, ZPZV<1>, ZPZV<1>, ZPZV<2»; }; // NOLINT
                                   template<> struct ConwayPolynomial<179, 13> { using ZPZ = aerobus::zpz<179>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<177»; }; // NOLINT</pre>
04866
                                   template<> struct ConwayPolynomial<179, 17> { using ZPZ = aerobus::zpz<179>; using type
                     POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<177»; }; //</pre>
                     NOLINT
04868
                                  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
 04870
                                 template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179»; }; // NOLINT template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
 04871
                    POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<10>, ZPZV<2); ; // NOLINT template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
 04872
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<179»; }; // NOLINT
 04873
                                 template<> struct ConwayPolynomial<181, 6> { using ZPZ = aerobus::zpz<181>; using type =
                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<163>, ZPZV<169>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type = POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4 (NOLINT)
 04874
```

```
template<> struct ConwayPolynomial<181, 8> { using ZPZ = aerobus::zpz<181>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<108>, ZPZV<22>, ZPZV<149>, ZPZV<2»; }; //
                       NOLINT
                      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>
04876
                       }; // NOLINT
                      template<> struct ConwayPolynomial<181, 10> { using ZPZ = aerobus::zpz<181>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<104>, ZPZV<104>, ZPZV<94>, ZPZV<57>, ZPZV<88>,
                       ZPZV<2»; }; // NOLINT</pre>
                      template<> struct ConwayPolynomial<181, 11> { using ZPZ = aerobus::zpz<181>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04878
                       ZPZV<24>, ZPZV<179»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<181, 12> { using ZPZ = aerobus::zpz<181>; using type =
04879
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<141>, ZPZV<45>, ZPZV<122>,
                       ZPZV<175>, ZPZV<12>, ZPZV<10>, ZPZV<2»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<181, 13> { using ZPZ = aerobus::zpz<181>; using type =
04880
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<179»; };</pre>
                                                                                                                                                                               // NOLINT
                                      template<> struct ConwayPolynomial<181, 17> { using ZPZ = aerobus::zpz<181>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<179»; };</pre>
04882
                                    template<> struct ConwayPolynomial<181, 19> { using ZPZ = aerobus::zpz<181>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<379»; }; //</pre>
                       NOLINT
                                      template<> struct ConwayPolynomial<191, 1> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<172»; }; // NOLINT
04884
                                     template<> struct ConwayPolynomial<191, 2> { using ZPZ = aerobus::zpz<191>; using type =
                       POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19»; }; // NOLINT
                                    template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
04885
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172»; }; // NOLINT
04886
                                      template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; }; // NOLINT
 04887
                                   template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<172»; }; // NOLINT template<> struct ConwayPolynomial<191, 6> { using ZPZ = aerobus::zpz<191>; using type =
04888
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<19); // NOLINT template<> struct ConwayPolynomial<191, 7> { using ZPZ = aerobus::zpz<191>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<14>, ZPZV<172»; };
                                   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 =
04891
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<124>, ZPZV<124>;
                                    template<> struct ConwayPolynomial<191, 10> { using ZPZ = aerobus::zpz<191>; using type :
04892
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<113>, ZPZV<47>, ZPZV<173>, ZPZV<74>,
                       ZPZV<156>, ZPZV<19»; }; // NOLINT
04893
                                    template<> struct ConwayPolynomial<191, 11> { using ZPZ = aerobus::zpz<191>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<6>, ZPZV<172»; };</pre>
                                                                                                                // NOLINT
                                     template<> struct ConwayPolynomial<191, 12> { using ZPZ = aerobus::zpz<191>; using type
 04894
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, 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 =
04895
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   template<> struct ConwayPolynomial<191,
                                                                                                                                                                                         17> { using ZPZ = aerobus::zpz<191>; using type =
 04896
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<172»; };</pre>
                                                                                                                                                                                                                                                                                                                    // NOLINT
                                     template<> struct ConwayPolynomial<191, 19> { using ZPZ = aerobus::zpz<191>; using type =
04897
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<190>, ZPZV<2>, ZPZV<172»; }; //</pre>
                                     template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<188»; }; // NOLINT
04899
                                     template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5»; }; // NOLINT
                                     template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
04900
                     POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
 04901
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<5»; }; // NOLINT
                                     template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
 04902
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<188»; }; // NOLINT
                                     template<> struct ConwayPolynomial<193, 6> { using ZPZ = aerobus::zpz<193>; using type =
04903
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<8>, ZPZV<172>, ZPZV<5»; }; // NOLINT
                                    template<> struct ConwayPolynomial<193, 7> { using ZPZ = aerobus::zpz<193>; using type
 04904
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 8> { using ZPZ = aerobus::zpz<193>; using type =
04905
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<145>, ZPZV<34>, ZPZV<154>, ZPZV<5»; }; //
                       NOLINT
                                     template<> struct ConwayPolynomial<193, 9> { using ZPZ = aerobus::zpz<193>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<168>, ZPZV<27>, ZPZV<188»;
                       }; // NOLINT
04907
                                     template<> struct ConwayPolynomial<193, 10> { using ZPZ = aerobus::zpz<193>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<51>, ZPZV<77>, ZPZV<0>, ZPZV<89>,
                       ZPZV<5»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type :
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; }; // NOLINT
                           template<> struct ConwayPolynomial<193, 12> { using ZPZ = aerobus::zpz<193>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<52>, ZPZV<135>, ZPZV<152>,
                            ZPZV<90>, ZPZV<46>, ZPZV<28>, ZPZV<5»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<193, 13> { using ZPZ = aerobus::zpz<193>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<188»; }; // NOLINT</pre>
04911
                                              template<> struct ConwayPolynomial<193, 17> { using ZPZ = aerobus::zpz<193>; using type =
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
04912
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<188»; };</pre>
                                              template<> struct ConwayPolynomial<197, 1> { using ZPZ = aerobus::zpz<197>; using type =
04913
                           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
                                               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 =
04916
                          POLYY<ZPZV<1>, ZPZV<1>, ZPZV<16>, ZPZV<124>, ZPZV<2; ; // NOLINT
template<> struct ConwayPolynomial<197, 5> { using ZPZ = aerobus::zpz<197>; using type =
04917
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195»; }; // NOLINT
 04918
                                             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<20; }; // NOLINT template<> struct ConwayPolynomial<197, 7> { using ZPZ = aerobus::zpz<197>; using type :
 04919
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                                            template<> struct ConwayPolynomial<197, 8> { using ZPZ = aerobus::zpz<197>; using type =
04920
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<176>, ZPZV<96>, ZPZV<29>, ZPZV<2»; };
                                           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<12>, ZPZV<127>, ZPZV<8>, ZPZV<195»;
                            }; // NOLINT
                                               template<> struct ConwayPolynomial<197, 10> { using ZPZ = aerobus::zpz<197>; using type =
04922
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<121>, ZPZV<137>, ZPZV<8>, ZPZV<73>, ZPZV<42>,
                            ZPZV<2»: }: // NOLINT
                                              template<> struct ConwayPolynomial<197, 11> { using ZPZ = aerobus::zpz<197>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<14>, ZPZV<195»; }; // NOLINT
    template<> struct ConwayPolynomial<197, 12> { using ZPZ = aerobus::zpz<197>; using type =
04924
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<20, ZPZV<0>, ZPZV<0>, ZPZV<3, ZPZV<168>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>, ZPZV<90, ZPZV<90, ZPZV<163>, ZPZV<2»; }; // NOLINT
                                              template<> struct ConwayPolynomial<197, 13> { using ZPZ = aerobus::zpz<197>; using type
04925
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<195»; }; // NOLINT
template<> struct ConwayPolynomial<197, 17> { using ZPZ = aerobus::zpz<197>; using type =
04926
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<195»; }; // NOLINT
template<> struct ConwayPolynomial<197, 19> { using ZPZ = aerobus::zpz<197>; using type
 04927
                            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<6>, ZPZV<195»; }; //</pre>
                            NOLINT
04928
                                              template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
                            POLYV<ZPZV<1>, ZPZV<196»; }; // NOLINT
                                            template<> struct ConwayPolynomial<199, 2> { using ZPZ = aerobus::zpz<199>; using type =
 04929
                           POLYV<ZPZV<1>, ZPZV<193>, ZPZV<3»; }; // NOLINT
 04930
                                               template<> struct ConwayPolynomial<199, 3> { using ZPZ = aerobus::zpz<199>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; }; // NOLINT template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
04931
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; };
                                                                                                                                                                                                                                                                                           // NOLINT
                                               template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<196»; }; // NOLINT
 04933
                                            template<> struct ConwayPolynomial<199, 6> { using ZPZ = aerobus::zpz<199>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<90>, ZPZV<58>, ZPZV<79>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<199, 7> { using ZPZ = aerobus::zpz<199>; using type =
04934
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
                                               template<> struct ConwayPolynomial<199, 8> { using ZPZ = aerobus::zpz<199>; using type
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<160>, ZPZV<23>, ZPZV<159>, ZPZV<3»; };
                          template<> struct ConwayPolynomial<199, 9> { using ZPZ = aerobus::zpz<199>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<177>, ZPZV<141>, ZPZV<141>, ZPZV<196»;</pre>
04936
                           }; // NOLINT
  template<> struct ConwayPolynomial<199, 10> { using ZPZ = aerobus::zpz<199>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<158>, ZPZV<31>, ZPZV<54>, ZPZV<9>,
                             ZPZV<3»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<199, 11> { using ZPZ = aerobus::zpz<199>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<1>, ZPZV<196»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<199, 12> { using ZPZ = aerobus::zpz<199>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<33>, ZPZV<192>, ZPZV<197>, ZPZV<138>,
                            ZPZV<69>, ZPZV<57>, ZPZV<151>, ZPZV<3»; }; // NOLINT</pre>
04940
                                            template<> struct ConwayPolynomial<199, 13> { using ZPZ = aerobus::zpz<199>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
```

```
template<> struct ConwayPolynomial<199, 17> { using ZPZ = aerobus::zpz<199>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             \texttt{ZPZV} < \texttt{0>, ZPZV} < \texttt{13>, ZPZV} < \texttt{196} *; }; \\
                                                                                                                                                                                                                                                                                                                                                                                                       // NOLINT
                           template<> struct ConwayPolynomial<199, 19> { using ZPZ = aerobus::zpz<199>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04942
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<196*; }; //</pre>
                            NOLINT
04943
                                               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 =
04944
                           POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; }; // NOLINT
                                              template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
04945
                          POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2O9»; }; // NOLINT template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
04946
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2»; }; // NOLINT
                         template<> struct ConwayPolynomial<211, 5> { using ZPZ = aerobus::zpz<211>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<209»; }; // NOLINT</pre>
04947
                                              template<> struct ConwayPolynomial<211, 6> { using ZPZ = aerobus::zpz<211>; using type =
04948
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<81>, ZPZV<194>, ZPZV<133>, ZPZV<2»; }; // NOLINT
                                             template<> struct ConwayPolynomial<211, 7> { using ZPZ = aerobus::zpz<211>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<209»; }; //
04950
                                             template<> struct ConwayPolynomial<211, 8> { using ZPZ = aerobus::zpz<211>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<20>, ZPZV<87>, ZPZV<29>, ZPZV<2»; };
                            NOLINT
04951
                                              template<> struct ConwayPolynomial<211, 9> { using ZPZ = aerobus::zpz<211>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<139>, ZPZV<26-, ZPZV<209»;
                            }; // NOLINT
04952
                                            template<> struct ConwayPolynomial<211, 10> { using ZPZ = aerobus::zpz<211>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<61>, ZPZV<148>, ZPZV<148>, ZPZV<17, ZPZV<125>,
                            ZPZV<2»: }: // NOLINT</pre>
                                             template<> struct ConwayPolynomial<211, 11> { using ZPZ = aerobus::zpz<211>; using type =
04953
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<7>, ZPZV<209»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<50>, ZPZV<145>, ZPZV<126>, ZPZV<184>, ZPZV<84>, ZPZV<27>, ZPZV<284>, ZPZV<27>, ZPZV<284>, ZPZV<284
                                              template<> struct ConwayPolynomial<211, 13> { using ZPZ = aerobus::zpz<211>; using type =
04955
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<209»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<211, 17> { using ZPZ = aerobus::zpz<211>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<209»; }; // NOLINT
    template<> struct ConwayPolynomial<211, 19> { using ZPZ = aerobus::zpz<211>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
04957
                             ZPZV<0>, ZPZV<17>, ZPZV<209»; }; //</pre>
                            NOLINT
04958
                                             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 =
04959
                           POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; }; // NOLINT
                                              template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
04960
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<220»; }; // NOLINT
                                             template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
04961
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<63>, ZPZV<163>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<223, 5> { using ZPZ = aerobus::zpz<223>; using type =
04962
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; }; // NOLINT
                                               template<> struct ConwayPolynomial<223, 6> { using ZPZ = aerobus::zpz<223>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68>, ZPZV<24>, ZPZV<196>, ZPZV<3»; }; // NOLINT
                                              template<> struct ConwayPolynomial<223, 7> { using ZPZ = aerobus::zpz<223>; using type =
04964
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
04965
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<3»; }; //
                           NOLINT
                                             template<> struct ConwayPolynomial<223, 9> { using ZPZ = aerobus::zpz<223>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<164>, ZPZV<64>, ZPZV<220»;
                            }; // NOLINT
04967
                           template<> struct ConwayPolynomial<223, 10> { using ZPZ = aerobus::zpz<223>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<118>, ZPZV<177>, ZPZV<87>, ZPZV<99>, ZPZV<62>, ZPZV<62>, ZPZV<10+, Z
                            ZPZV<3»; }; // NOLINT</pre>
                                               template<> struct ConwayPolynomial<223, 11> { using ZPZ = aerobus::zpz<223>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<220»; }; // NOLINT
                           template<> struct ConwayPolynomial<223, 12> { using ZPZ = aerobus::zpz<223>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64>, ZPZV<94>, ZPZV<11>, ZPZV<105>, ZPZV<64>,
04969
                            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</pre>
                                               template<> struct ConwayPolynomial<223, 17> { using ZPZ = aerobus::zpz<223>; using type =
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>
                                             template<> struct ConwayPolynomial<227, 1> { using ZPZ = aerobus::zpz<227>; using type =
                            POLYV<ZPZV<1>, ZPZV<225»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<227, 2> { using ZPZ = aerobus::zpz<227>; using type =
                     POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2»; }; // NOLINT
04975
                                  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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2»; }; // NOLINT
 04976
                                   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
 04978
                                  template<> struct ConwayPolynomial<227, 6> { using ZPZ = aerobus::zpz<227>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<24>, ZPZV<135>, ZPZV<2»; }; // NOLINT
                                   template<> struct ConwayPolynomial<227, 7> { using ZPZ = aerobus::zpz<227>; using type =
04979
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<225»; }; // NOLINT
                                   template<> struct ConwayPolynomial<227, 8> { using ZPZ = aerobus::zpz<227>; using type =
04980
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<151>, ZPZV<176>, ZPZV<106>, ZPZV<2»; }; //
                      NOLINT
04981
                                  template<> struct ConwayPolynomial<227, 9> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<24>, ZPZV<24>, ZPZV<183>, ZPZV<225»;
                      }; // 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<199>, ZPZV<12>, ZPZV<12>, ZPZV<93>, ZPZV<77>,
                      ZPZV<2»; }; // NOLINT</pre>
04983
                                   template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type :
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04984
                                   template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<2PZV<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<2*; }; // NOLINT
                                  template<> struct ConwayPolynomial<227, 13> { using ZPZ = aerobus::zpz<227>; using type =
04985
                      \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                      ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<225»; }; // NOLINT</pre>
                                  template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type =
04986
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      template<> struct ConwayPolynomial<227, 19> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                                                                                                                                                                                                                                                                                                                                     7.P7.V<0>.
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3+, ZPZV<34</pre>
                      NOLINT
                                   template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
                     POLYV<ZPZV<1>, ZPZV<223»; }; // NOLINT
                                  template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
                    POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
04990
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; }; // NOLINT template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
 04991
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<6»; }; // NOLINT
 04992
                                   template<> struct ConwayPolynomial<229, 5> { using ZPZ = aerobus::zpz<229>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223»; }; // NOLINT
                    template<> struct ConwayPolynomial<229, 6> { using ZPZ = aerobus::zpz<229>; using type =
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 =
04993
04994
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<23»; }; //
                                   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»; }; //
                                  template<> struct ConwayPolynomial<229, 9> { using ZPZ = aerobus::zpz<229>; using type =
04996
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<117>, ZPZV<50>, ZPZV<223»;
 04997
                                    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 =
04998
                      \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, "ZPZV<0>, "ZPZV<0>,
                      ZPZV<2>, ZPZV<223»; }; // NOLINT</pre>
                                   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</pre>
                                    template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type :
05000
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<223»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<229,
                                                                                                                                                                               17> { using ZPZ = aerobus::zpz<229>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      // NOLINT
                     template<> struct ConwayPolynomial<229, 19> { using ZPZ = aerobus::zpz<229>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05002
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<228>, ZPZV<15>, ZPZV<223»; }; //</pre>
 05003
                                  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 =
05004
                     POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
                                   template<> struct ConwayPolynomial<233, 3> { using ZPZ = aerobus::zpz<233>; using type =
 05005
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<3»; }; // NOLINT
 05007
                                 template<> struct ConwayPolynomial<233, 5> { using ZPZ = aerobus::zpz<233>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<230»; }; // NOLINT
 05008
                                  template<> struct ConwayPolynomial<233, 6> { using ZPZ = aerobus::zpz<233>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<122>, ZPZV<215>, ZPZV<32>, ZPZV<3»; };
                                            template<> struct ConwayPolynomial<233, 7> { using ZPZ = aerobus::zpz<233>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230»; }; // NOLINT
 05010
                                         template<> struct ConwayPolynomial<233, 8> { using ZPZ = aerobus::zpz<233>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<202>, ZPZV<135>, ZPZV<181>, ZPZV<3»; }; //
                                            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
05012
                                            template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<28>, ZPZV<71>, ZPZV<102>, ZPZV<48>,
                           ZPZV<3»: }: // NOLINT
05013
                                            template<> struct ConwayPolynomial<233, 11> { using ZPZ = aerobus::zpz<233>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<5>, ZPZV<230»; }; // NOLINT</pre>
05014
                                          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<96>, ZPZV<21>, ZPZV<114>, ZPZV<11-, ZPZV<
                           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
                           ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<230»; }; // NOLINT</pre>
05016
                                         template<> struct ConwayPolynomial<233, 17> { using ZPZ = aerobus::zpz<233>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<25>, ZPZV<230»; }; //</pre>
                           NOLINT
05018
                                           template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
                          POLYV<ZPZV<1>, ZPZV<232»; }; // NOLINT
                                          template<> struct ConwayPolynomial<239, 2> { using ZPZ = aerobus::zpz<239>; using type =
05019
                          POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7»; }; // NOLINT
                                            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 =
 05021
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<239, 5> { using ZPZ = aerobus::zpz<239>; using type =
05022
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; }; // NOLINT
 05023
                                            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 =
 05024
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2
05025
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<202>, ZPZV<54>, ZPZV<5*, }; //
 05026
                                          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<3232»; };
                           // NOLINT
05027
                                           template<> struct ConwavPolynomial<239, 10> { using ZPZ = aerobus::zpz<239>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<68>, ZPZV<226>, ZPZV<127>,
                           ZPZV<108>, ZPZV<7»; };</pre>
                                                                                                                                   // NOLINT
                                          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 =
05029
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>, ZPZV<14>, ZPZV<113>, ZPZV<182>, ZPZV<101>, ZPZV<81>, ZPZV<216>, ZPZV<7»; }; // NOLINT
                                         template<> struct ConwayPolynomial<239, 13> { using ZPZ = aerobus::zpz<239>; using type
 05030
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                          template<> struct ConwayPolynomial<239, 17> { using ZPZ = aerobus::zpz<239>; using type =
05031
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<232»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                            // NOLINT
                                            template<> struct ConwayPolynomial<239, 19> { using ZPZ = aerobus::zpz<239>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<24>, ZPZV<24>, ZPZV<232»; }; //</pre>
                           NOLINT
                                           template<> struct ConwayPolynomial<241, 1> { using ZPZ = aerobus::zpz<241>; using type =
05033
                          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
 05035
                                         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 =
 05036
                         POLYVCZPZV<1>, ZPZV<1>, ZPZV<14>, ZPZV<152>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<241, 5> { using ZPZ = aerobus::zpz<241>; using type =
                          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
05039
                                          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<234w; }; // NOLINT
                                            template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<113>, ZPZV<212>, ZPZV<153>, ZPZV<17»; }; //
                           NOLINT
05041
                                         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<1255, ZPZV<234»;
                           }; // 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<268>, ZPZV<55>,
                      ZPZV<7»; }; // NOLINT</pre>
05043
                                  template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                             // NOLINT
                      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</pre>
05045
                                   template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    template<> struct ConwayPolynomial<241,
                                                                                                                                                                                    17> { using ZPZ = aerobus::zpz<241>; using type
05046
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      template<> struct ConwayPolynomial<241, 19> { using ZPZ = aerobus::zpz<241>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<234»; }; //
05047
                                     template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
                      POLYV<ZPZV<1>, ZPZV<245»; }; // NOLINT
05049
                                    template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
                     POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
05050
                     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
                                   template<> struct ConwayPolynomial<251, 5> { using ZPZ = aerobus::zpz<251>; using type =
05052
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT
                                   template<> struct ConwayPolynomial<251, 6> { using ZPZ = aerobus::zpz<251>; using type =
05053
                    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
05054
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<245»; }; //
05055
                                 template<> struct ConwayPolynomial<251, 8> { using ZPZ = aerobus::zpz<251>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<142>, ZPZV<215>, ZPZV<173>, ZPZV<6»; }; //
                      NOLINT
                                   template<> struct ConwayPolynomial<251, 9> { using ZPZ = aerobus::zpz<251>; using type =
05056
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<106>, ZPZV<245»;
                                     template<> struct ConwayPolynomial<251, 10> { using ZPZ = aerobus::zpz<251>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<138>, ZPZV<110>, ZPZV<45>, ZPZV<34>,
                      ZPZV<149>, ZPZV<6»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type =
05058
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<26>, ZPZV<245»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type
05059
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<192>, ZPZV<53>, ZPZV<20>, ZPZV<20>, ZPZV<15>,
                      \mbox{ZPZV}\mbox{<201>, ZPZV}\mbox{<232>, ZPZV}\mbox{<6}\mbox{*; }; \mbox{$//$ NOLINT$}
05060
                                   template<> struct ConwayPolynomial<251, 13> { using ZPZ = aerobus::zpz<251>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<245»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<251, 17> { using ZPZ = aerobus::zpz<251>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     05062
                      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>
                                     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 =
05064
                     POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3»; }; // NOLINT
05065
                                     template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
                     POLYY<ZPZY<1>, ZPZY<0>, ZPZY<6>, ZPZY<65, ZPZY<254»; }; // NOLINT template<> struct ConwayPolynomial<257, 4> { using ZPZ = aerobus::zpz<257>; using type =
05066
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<187>, ZPZV<3»; }; // NOLINT
                                    template<> struct ConwayPolynomial<257, 5> { using ZPZ = aerobus::zpz<257>; using type =
05067
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<254»; }; // NOLINT
                                   template<> struct ConwayPolynomial<257, 6> { using ZPZ = aerobus::zpz<257>; using type =
05068
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<62>, ZPZV<18>, ZPZV<138>, ZPZV<3»; };
                                   template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<31>, ZPZV<254»; };
05070
                                   template<> struct ConwayPolynomial<257, 8> { using ZPZ = aerobus::zpz<257>; using type =
                     POLYV<2PZV<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 =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<201>, ZPZV<50>, ZPZV<254»;
                      }; // NOLINT
05072
                                    template<> struct ConwayPolynomial<257, 10> { using ZPZ = aerobus::zpz<257>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<97>, ZPZV<12>, ZPZV<225>, ZPZV<180>, ZPZV<20>,
                      ZPZV<3»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<257, 11> { using ZPZ = aerobus::zpz<257>; using type :
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<40>, ZPZV<254»; };</pre>
                                                                                                           // NOLINT
05074
                                   template<> struct ConwayPolynomial<257, 12> { using ZPZ = aerobus::zpz<257>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<13>, ZPZV<225>, ZPZV<215>, ZPZV<2173>, ZPZV<249>, ZPZV<148>, ZPZV<20>, ZPZV<3»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<257, 13> { using ZPZ = aerobus::zpz<257>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                template<> struct ConwayPolynomial<257, 17> { using ZPZ = aerobus::zpz<257>; using type =
05076
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<254»; };</pre>
                                                                                                                                                                                                                                                                                                 // NOLINT
                                  template<> struct ConwayPolynomial<257, 19> { using ZPZ = aerobus::zpz<257>; using type
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<254»; }; //</pre>
                     NOLINT
05078
                                 template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
                    POLYV<ZPZV<1>, ZPZV<258»; }; // NOLINT
                                  template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
05079
                    POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; }; // NOLINT
05080
                                template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; }; // NOLINT template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; }; // NOLINT
05081
                                  template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
05082
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<258»; }; // NOLINT
                                   template<> struct ConwayPolynomial<263, 6> { using ZPZ = aerobus::zpz<263>; using type =
                    POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<25>, ZPZV<225>, ZPZV<5»; }; // NOLINT
05084
                                 template<> struct ConwayPolynomial<263, 7> { using ZPZ = aerobus::zpz<263>; using type =
                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05085
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<170>, ZPZV<7>, ZPZV<5»; };
                                template<> struct ConwayPolynomial<263, 9> { using ZPZ = aerobus::zpz<263>; using type =
05086
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<261>, ZPZV<261>, ZPZV<29>, ZPZV<258»;
                     }; // NOLINT
                                  template<> struct ConwayPolynomial<263, 10> { using ZPZ = aerobus::zpz<263>; using type =
05087
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<245>, ZPZV<231>, ZPZV<198>, ZPZV<145>,
                     ZPZV<119>, ZPZV<5»; };
                                                                                                   // NOLINT
                                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 =
05089
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<174>, ZPZV<162>, ZPZV<252>,
                     ZPZV<47>, ZPZV<45>, ZPZV<180>, ZPZV<5»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
05090
                    POLYV<ZPZV<1>, ZPZV<267»; }; // NOLINT
                                  template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
05091
                     POLYV<ZPZV<1>. ZPZV<268>. ZPZV<2»: }: // NOLINT
                                  template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
05092
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<267»; }; // NOLINT
                                  template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
05093
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2.; }; // NOLINT template<> struct ConwayPolynomial<269, 5> { using ZPZ = aerobus::zpz<269>; using type =
05094
                    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 =
05095
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<101>, ZPZV<206>, ZPZV<2»; }; // NOLINT
                                  template<> struct ConwayPolynomial<269, 7> { using ZPZ = aerobus::zpz<269>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 - , ZPZ
05097
                                 template<> struct ConwayPolynomial<269, 8> { using ZPZ = aerobus::zpz<269>; using type =
                     POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<220>, ZPZV<131>, ZPZV<232>, ZPZV<23; }; //
                     NOLINT
                                   template<> struct ConwayPolynomial<269, 9> { using ZPZ = aerobus::zpz<269>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2+, ZPZV<214>, ZPZV<267>, ZPZV<267>;
                     }; // NOLINT
05099
                                  template<> struct ConwayPolynomial<269, 10> { using ZPZ = aerobus::zpz<269>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<264>, ZPZV<243>, ZPZV<186>, ZPZV<61>, ZPZV<10>, ZPZV<20>; // NOLINT
                                  template<> struct ConwayPolynomial<269, 11> { using ZPZ = aerobus::zpz<269>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                     ZPZV<20>, ZPZV<267»; };</pre>
                                                                                                      // NOLINT
                    template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<165>, ZPZV<63>, ZPZV<215>,
05101
                     ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type =
05102
                    POLYV<ZPZV<1>, ZPZV<265»; }; // NOLINT
                                  template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
05103
                    POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; }; // NOLINT
                                  template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
05104
                   POLYV<ZPZV<1>, ZPZV<0, ZPZV<2>, ZPZV<265»; }; // NOLINT
template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6»; }; // NOLINT
05105
                                 template<> struct ConwayPolynomial<271, 5> { using ZPZ = aerobus::zpz<271>; using type =
05106
                    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 =
05107
                   POLYV<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 = aerobus::zpz<2
05108
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<265»; }; // NOLINT
                                 template<> struct ConwayPolynomial<271, 8> { using ZPZ = aerobus::zpz<271>; using type
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<114>, ZPZV<69>, ZPZV<68>; };
                    template<> struct ConwayPolynomial<271, 9> { using ZPZ = aerobus::zpz<271>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<266>, ZPZV<186>, ZPZV<266>;
05110
```

```
}; // NOLINT
                   template<> struct ConwayPolynomial<271, 10> { using ZPZ = aerobus::zpz<271>; using type =
05111
           POLYV<ZPZV<1>, ZPZV<0>, ZPŽV<0>, ZPZV<0>, ZPZV<74>, ZPZV<133>, ZPZV<10>, ZPZV<256>, ZPŽV<74>,
           ZPZV<126>, ZPZV<6»; }; // NOLINT</pre>
                   template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type :
05112
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                   template<> struct ConwayPolynomial<271, 12> { using ZPZ = aerobus::zpz<271>; using type
05113
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162>, ZPZV<210>, ZPZV<116>, ZPZV<205>, ZPZV<237>, ZPZV<256>, ZPZV<130>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
05114
          POLYV<ZPZV<1>, ZPZV<272»; }; // NOLINT
                   template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
05115
           POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; }; // NOLINT
05116
                  template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; }; // NOLINT template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
05117
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, 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
                   template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type =
0.5119
          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 =
05120
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>; ; // NOLINT template<> struct ConwayPolynomial<277, 8> { using ZPZ = aerobus::zpz<277>; using type =
05121
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<159>, ZPZV<176>, ZPZV<5»; }; //
                  template<> struct ConwayPolynomial<277, 9> { using ZPZ = aerobus::zpz<277>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<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>
                 template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<272»; }; // NOLINT
                  template<> struct ConwayPolynomial<277, 12> { using ZPZ = aerobus::zpz<277>; using type =
05125
           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>
                   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 =
05127
           POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»: }: // NOLINT
                  template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<278»; ); // NOLINT template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
05129
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<281, 5> { using ZPZ = aerobus::zpz<281>; using type =
05130
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<278»; }; // NOLINT
                   template<> struct ConwayPolynomial<281, 6> { using ZPZ = aerobus::zpz<281>; using type =
05131
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<15), ZPZV<13>, ZPZV<27>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<281, 7> { using ZPZ = aerobus::zpz<281>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<278»; };
05133
                  template<> struct ConwayPolynomial<281, 8> { using ZPZ = aerobus::zpz<281>; using type =
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195>, ZPZV<279>, ZPZV<140>, ZPZV<3»; }; //
           NOLINT
05134
                   template<> struct ConwayPolynomial<281, 9> { using ZPZ = aerobus::zpz<281>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<148>, ZPZV<70>, ZPZV<278»;
           }; // NOLINT
05135
                   template<> struct ConwayPolynomial<281, 10> { using ZPZ = aerobus::zpz<281>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<145>, ZPZV<13>, ZPZV<138>, ZPZV<191>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<281, 11> { using ZPZ = aerobus::zpz<281>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
           ZPZV<36>, ZPZV<278»; };</pre>
                                                        // NOLINT
          template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type = POLYV-ZPZV-1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<10>, ZPZV<16>, ZPZV<10>, ZPZV<16>, ZPZV<202>, ZPZV<202>, ZPZV<28>, ZPZV<103>, ZPZV<116>, ZPZV<58>, ZPZV<28>, ZPZV<291>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
05137
05138
          POLYV<ZPZV<1>, ZPZV<280»; }; // NOLINT
                  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 =
05140
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type =
05141
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
05142
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<280»; }; // NOLINT
05143
                   template<> struct ConwayPolynomial<283, 6> { using ZPZ = aerobus::zpz<283>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<68>, ZPZV<73>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type =
05144
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<80»; }; // NOLINT
                  template<> struct ConwayPolynomial<283, 8> { using ZPZ = aerobus::zpz<283>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<179>, ZPZV<32>, ZPZV<232>, ZPZV<23»; }; //
           template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<280»;</pre>
05146
```

```
}; // NOLINT
05147
                             template<> struct ConwayPolynomial<283, 10> { using ZPZ = aerobus::zpz<283>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<271>, ZPZV<185>, ZPZV<68>, ZPŽV<100>,
                 ZPZV<219>, ZPZV<3»; }; // NOLINT</pre>
                 template<> struct ConwayPolynomial<283, 11> { using ZPZ = aerobus::zpz<283>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
0.5148
                             template<> struct ConwayPolynomial<283, 12> { using ZPZ = aerobus::zpz<283>; using type :
05149
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<8>, ZPZV<96>, ZPZV<229>, ZPZV<49>, ZPZV<14>, ZPZV<56>, ZPZV<3»; }; // NOLINT
05150
                           template<> struct ConwayPolynomial<293, 1> { using ZPZ = aerobus::zpz<293>; using type =
                 POLYV<ZPZV<1>, ZPZV<291»; }; // NOLINT
                            template<> struct ConwayPolynomial<293, 2> { using ZPZ = aerobus::zpz<293>; using type =
05151
                 POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; }; // NOLINT
05152
                           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 =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
05153
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
                             template<> struct ConwayPolynomial<293, 6> { using ZPZ = aerobus::zpz<293>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<210>, ZPZV<260>, ZPZV<2»; }; // NOLINT
05156
                           template<> struct ConwayPolynomial<293, 7> { using ZPZ = aerobus::zpz<293>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
05157
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<2»; }; //
                template<> struct ConwayPolynomial<293, 9> { using ZPZ = aerobus::zpz<293>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<208>, ZPZV<190>, ZPZV<291»;</pre>
05158
                 }; // 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<0
                 ZPZV<3>, ZPZV<291»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<293, 12> { using ZPZ = aerobus::zpz<293>; using type =
05161
                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2159>, ZPZV<210>, ZPZV<125>, ZPZV<212>,
                  ZPZV<167>, ZPZV<144>, ZPZV<157>, ZPZV<2»; }; // NOLINT</pre>
                            template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
05162
                 POLYV<ZPZV<1>, ZPZV<302»; }; // NOLINT
                            template<> struct ConwayPolynomial<307, 2> { using ZPZ = aerobus::zpz<307>; using type =
0.5163
                 POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5»; }; // NOLINT
                            template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
05164
                POLYV<ZPZV<1>, ZPZV<2>, ZPZV<3>, ZPZV<302»; }; // NOLINT template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
05165
                POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<307, 5> { using ZPZ = aerobus::zpz<307>; using type =
05166
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<302»; }; // NOLINT
                            template<> struct ConwayPolynomial<307, 6> { using ZPZ = aerobus::zpz<307>; using type =
05167
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<213>, ZPZV<172>, ZPZV<61>, ZPZV<5»; }; // NOLINT
                            template<> struct ConwayPolynomial<307, 7> { using ZPZ = aerobus::zpz<307>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 - , Z
05169
                           template<> struct ConwayPolynomial<307, 8> { using ZPZ = aerobus::zpz<307>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<283>, ZPZV<232>, ZPZV<131>, ZPZV<5»; }; //
                 NOLINT
05170
                            template<> struct ConwayPolynomial<307, 9> { using ZPZ = aerobus::zpz<307>; using type =
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<165>, ZPZV<70>, ZPZV<302»;
                 }; // NOLINT
05171
                             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 =
05172
                POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; }; // NOLINT
                            template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type = \(^1\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\(^2\)\
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; };
                           template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
05174
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
05175
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294»; }; // NOLINT
05176
                             template<> struct ConwayPolynomial<311, 6> { using ZPZ = aerobus::zpz<311>; using type =
                 POLYV<2PZV<1>, 2PZV<0>, 2PZV<1>, 2PZV<27>, ZPZV<167>, ZPZV<152>, ZPZV<17»; }; // NOLINT
05177
                          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<1>), ZPZV<294»; ); // NOLINT template<> struct ConwayPolynomial<311, 8> { using ZPZ = aerobus::zpz<311>; using type =
05178
                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<162>, ZPZV<118>, ZPZV<2>, ZPZV<217»; }; //
                           template<> struct ConwayPolynomial<311, 9> { using ZPZ = aerobus::zpz<311>; using type =
05179
                 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
05180
                            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 =
                 POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; }; // NOLINT
05182
                           template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; }; // NOLINT template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<239>, ZPZV<10»; }; // NOLINT
05183
```

```
05184
            template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<303»; }; // NOLINT
           template<> struct ConwayPolynomial<313, 6> { using ZPZ = aerobus::zpz<313>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<196>, ZPZV<213>, ZPZV<25>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<313, 7> { using ZPZ = aerobus::zpz<313>; using type
05186
      POLYV-ZPZV-1>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-3); // NoLII template<> struct ConwayPolynomial<313, 8> { using ZPZ = aerobus::zpz<313>; using type :
                                                                                                          // NOLINT
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<306>, ZPZV<99>, ZPZV<106>, ZPZV<10»; };
       template<> struct ConwayPolynomial<313, 9> { using ZPZ = aerobus::zpz<313>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<267>, ZPZV<300>, ZPZV<303»;</pre>
05188
       }; // NOLINT
05189
            template<> struct ConwayPolynomial<317, 1> { using ZPZ = aerobus::zpz<317>; using type =
       POLYV<ZPZV<1>, ZPZV<315»; // NOLINT
05190
           template<> struct ConwayPolynomial<317, 2> { using ZPZ = aerobus::zpz<317>; using type =
       POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
05191
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<315»; };
                                                                // NOLINT
            template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<317, 5> { using ZPZ = aerobus::zpz<317>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<315»; // NOLINT
           template<> struct ConwayPolynomial<317, 6> { using ZPZ = aerobus::zpz<317>; using type =
0.5194
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<195>, ZPZV<156>, ZPZV<4>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<317, 7> { using ZPZ = aerobus::zpz<317>; using type
05195
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<315»; }; //
05196
           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
05197
           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
  template<> struct ConwayPolynomial<331, 1> { using ZPZ = aerobus::zpz<331>; using type =
05198
       POLYV<ZPZV<1>, ZPZV<328»; }; // NOLINT
05199
            template<> struct ConwayPolynomial<331, 2> { using ZPZ = aerobus::zpz<331>; using type =
       POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
05200
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<328»; }; // NOLINT
05201
            template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<2>, ZPZV<29>, ZPZV<290>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<331, 5> { using ZPZ = aerobus::zpz<331>; using type =
05202
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<328»; }; // NOLINT template<> struct ConwayPolynomial<331, 6> { using ZPZ = aerobus::zpz<331>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<205>, ZPZV<159>, ZPZV<3»; }; // NOLINT
05203
            template<> struct ConwayPolynomial<331, 7> { using ZPZ = aerobus::zpz<331>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<328»; };
05205
           template<> struct ConwayPolynomial<331, 8> { using ZPZ = aerobus::zpz<331>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<249>, ZPZV<308>, ZPZV<78>, ZPZV<3»; }; //
       NOLINT
           template<> struct ConwayPolynomial<331, 9> { using ZPZ = aerobus::zpz<331>; using type =
05206
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<194>, ZPZV<210>, ZPZV<328»;
       }; // NOLINT
05207
            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 =
05208
       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 =
05210
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<337, 5> { using ZPZ = aerobus::zpz<337>; using type =
05211
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT
05212
            template<> struct ConwayPolynomial<337, 6> { using ZPZ = aerobus::zpz<337>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<216>, ZPZV<127>, ZPZV<109>, ZPZV<10»; }; // NOLINT
05213
           template<> struct ConwayPolynomial<337, 7> { using ZPZ = aerobus::zpz<337>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<327*; }; // NOLINT template<> struct ConwayPolynomial<337, 8> { using ZPZ = aerobus::zpz<337>; using type =
05214
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<331>, ZPZV<246>, ZPZV<251>, ZPZV<10»; }; //
       NOLINT
            template<> struct ConwayPolynomial<337, 9> { using ZPZ = aerobus::zpz<337>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<148>, ZPZV<98>, ZPZV<327»;
       }; // NOLINT
05216
           template<> struct ConwayPolynomial<347, 1> { using ZPZ = aerobus::zpz<347>; using type =
       POLYV<ZPZV<1>, ZPZV<345»; }; // NOLINT
           template<> struct ConwayPolynomial<347, 2> { using ZPZ = aerobus::zpz<347>; using type =
05217
       POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
05218
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
05219
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
05220
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<345»; }; // NOLINT
            template<> struct ConwayPolynomial<347, 6> { using ZPZ = aerobus::zpz<347>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<343>, ZPZV<26>, ZPZV<56>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
05222
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3, ZPZV<4, ZPZV<45, ZPZV<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type =
05223
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<187>, ZPZV<213>, ZPZV<117>, ZPZV<2*; }; //
05224
           template<> struct ConwayPolynomial<347, 9> { using ZPZ = aerobus::zpz<347>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<235>, ZPZV<252>, ZPZV<252>, ZPZV<345»;
       }; // NOLINT
05225
           template<> struct ConwavPolynomial<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 =
05226
       POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; }; // NOLINT
05227
           template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; }; // NOLINT template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
05228
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2w; };
                                                                        // NOLINT
           template<> struct ConwayPolynomial<349, 5> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<347»; }; // NOLINT
05230
           template<> struct ConwayPolynomial<349, 6> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<135>, ZPZV<177>, ZPZV<316>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<349, 7> { using ZPZ = aerobus::zpz<349>; using type
05231
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347»; }; // NOLINT
           template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type =
05232
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<328>, ZPZV<268>, ZPZV<28; };
       NOLINT
05233
           template<> struct ConwayPolynomial<349, 9> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<290>, ZPZV<130>, ZPZV<347»;
       }; // NOLINT
            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 =
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
05236
      POLYV<ZPZV<1>, ZPZV<3>, ZPZV<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type =
05237
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<199>, ZPZV<3»; }; // NOLINT
          template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
05238
      05239
           template<> struct ConwayPolynomial<353, 6> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<215, ZPZV<26>, 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<0>, 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<182>, ZPZV<26>, ZPZV<37>, ZPZV<3»; };
       NOLINT
           template<> struct ConwayPolynomial<353, 9> { using ZPZ = aerobus::zpz<353>; using type =
05242
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<319>, ZPZV<49>, ZPZV<350»;
05243
           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 =
05244
      POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
05245
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<352,; }; // NOLINT template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
      05247
           template<> struct ConwayPolynomial<359, 5> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; }; // NOLINT
           template<> struct ConwayPolynomial<359, 6> { using ZPZ = aerobus::zpz<359>; using type =
05248
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<309>, ZPZV<327>, ZPZV<327>, ZPZV<7»; }; // NOLINT
           template<> struct ConwayPolynomial<359, 7> { using ZPZ = aerobus::zpz<359>; using type
05249
      POLYV<ZPZV<1>, 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 =
05250
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<301>, ZPZV<143>, ZPZV<271>, ZPZV<7»; }; //
       NOLINT
05251
           template<> struct ConwayPolynomial<359, 9> { using ZPZ = aerobus::zpz<359>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<356>, ZPZV<165>, ZPZV<352»;
       }; // NOLINT
05252
           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 =
05253
      POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6»; }; // NOLINT
           template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
05254
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<361»; }; // NOLINT template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type =
05255
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<367, 5> { using ZPZ = aerobus::zpz<367>; using type =
05256
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<35, ZPZV<35; }; // NOLINT template<> struct ConwayPolynomial<367, 6> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<321>, ZPZV<324>, ZPZV<6»; }; // NOLINT
05258
           template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type =
      template<> struct ConwayPolynomial<367, /> { using ZPZ = derobus::2p2<367>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<361»; }; // NOLINT template<> struct ConwayPolynomial<367, 8> { using ZPZ = derobus::2p2<367>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<282>, ZPZV<282>, ZPZV<50>, ZPZV<6»; };
05259
      template<> struct ConwayPolynomial<367, 9> { using ZPZ = aerobus::zpz<367>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<213>, ZPZV<213>, ZPZV<268>, ZPZV<361»;
       }; // NOLINT
           template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
05261
       POLYV<ZPZV<1>, ZPZV<371»; }; // NOLINT
```

```
05262
               template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
        POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2»; }; // NOLINT
05263
             template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<371»; }; // NOLINT
template<> struct ConwayPolynomial<373, 4> { using ZPZ = aerobus::zpz<373>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<373, 5> { using ZPZ = aerobus::zpz<373>; using type =
05264
        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 =
05266
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<83>, ZPZV<108>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type =
05267
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<371»; }; // NOLINT
              template<> struct ConwayPolynomial<373, 8> { using ZPZ = aerobus::zpz<373>; using type =
05268
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<203>, ZPZV<219>, ZPZV<66>, ZPZV<20; };
        NOLINT
              template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type =
05269
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<238>, ZPZV<370>, ZPZV<371»;
        }; // NOLINT
               template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
        POLYV<ZPZV<1>, ZPZV<377»; }; // NOLINT
               template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
        POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2»; }; // NOLINT
              \texttt{template<>} \texttt{struct ConwayPolynomial<379, 3> \{ \texttt{using ZPZ = aerobus::zpz<379>; 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
05272
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377»; }; // NOLINT template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
05273
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327>, ZPZV<2»; }; // NOLINT
05274
              template<> struct ConwayPolynomial<379, 5> { using ZPZ = aerobus::zpz<379>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<377»; }; // NOLINT
05275
              template<> struct ConwayPolynomial<379, 6> { using ZPZ = aerobus::zpz<379>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<374>, ZPZV<364>, ZPZV<246>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<379, 7> { using ZPZ = aerobus::zpz<379>; using type
05276
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, 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
              template<> struct ConwayPolynomial<379, 9> { using ZPZ = aerobus::zpz<379>; using type =
05278
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<36>, ZPZV<364, ZPZV<377»;
        }; // NOLINT
05279
               template<> struct ConwayPolynomial<383, 1> { using ZPZ = aerobus::zpz<383>; using type =
        POLYV<ZPZV<1>, ZPZV<378»; }; // NOLINT
05280
             template<> struct ConwayPolynomial<383, 2> { using ZPZ = aerobus::zpz<383>; using type =
        POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
05281
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
              template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5»; }; // NOLINT
05283
             template<> struct ConwayPolynomial<383, 5> { using ZPZ = aerobus::zpz<383>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT
              template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
05284
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<8>, ZPZV<158>, ZPZV<5»; }; // NOLINT
05285
              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<6>, ZPZV<6>, ZPZV<6>, ZPZV<378»; }; //
             template<> struct ConwayPolynomial<383, 8> { using ZPZ = aerobus::zpz<383>; using type
05286
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<281>, ZPZV<332>, ZPZV<296>, ZPZV<5»; }; //
        NOLINT
05287
              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
              template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
05288
        POLYV<ZPZV<1>, ZPZV<387»; }; // NOLINT
             template<> struct ConwayPolynomial<389, 2> { using ZPZ = aerobus::zpz<389>; using type =
05289
        POLYV<ZPZV<1>, ZPZV<379>, ZPZV<2»; }; // NOLINT
05290
              template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387»; }; // NOLINT template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
05291
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<266>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
05292
        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 =
05293
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<339>, ZPZV<255>, ZPZV<25; }; // NOLINT
              template<> struct ConwayPolynomial<389, 7> { using ZPZ = aerobus::zpz<389>; using type
05294
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<24>, ZPZV<24>, ZPZV<387»; };
05295
              template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::zpz<389>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35, ZPZV<19>, ZPZV<290>, ZPZV<2»; };
        NOLINT
              template<> struct ConwayPolynomial<389, 9> { using ZPZ = aerobus::zpz<389>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<308>, ZPZV<387»;
        }; // NOLINT
05297
              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 =
05298
        POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<397, 3> { using ZPZ = aerobus::zpz<397>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<392»; };
                                                                            // NOLINT
             template<> struct ConwayPolynomial<397, 4> { using ZPZ = aerobus::zpz<397>; using type =
05300
        POLYV<ZPZV<1>, ZPZV<3>, ZPZV<12>, ZPZV<363>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
05301
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392»; };
         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 =
05303
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<392»; }; // NOLINT template<> struct ConwayPolynomial<397, 8> { using ZPZ = aerobus::zpz<397>; using type =
05304
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<375>, ZPZV<255>, ZPZV<203>, ZPZV<5»; }; //
05305
               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»;
         }; // NOLINT
05306
               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 =
         POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3»; }; // NOLINT
05308
               template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<398»; }; // NOLINT template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<3»; }; // NOLINT
05309
               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
05311
               template<> struct ConwayPolynomial<401, 6> { using ZPZ = aerobus::zpz<401>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<115>, ZPZV<81>, ZPZV<51>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<401, 7> { using ZPZ = aerobus::zpz<401>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<398»; }; // NOLINT
05312
               template<> struct ConwayPolynomial<401, 8> { using ZPZ = aerobus::zpz<401; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<380>, ZPZV<113>, ZPZV<164>, ZPZV<3»; }; //
         NOLINT
         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>
05314
         }; // NOLINT
05315
                template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
         POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
              template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
05316
         POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
05317
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388»; }; // NOLINT
               template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<407>, ZPZV<21»; }; // NOLINT
05319
               template<> struct ConwayPolynomial<409, 5> { using ZPZ = aerobus::zpz<409>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<388»; }; // NOLINT
05320
               template<> struct ConwayPolynomial<409, 6> { using ZPZ = aerobus::zpz<409>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<372>, ZPZV<53>, ZPZV<364>, ZPZV<21»; }; // NOLINT
05321
               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»; }; //
05322
               template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<56>, ZPZV<69>, ZPZV<396>, ZPZV<31»; }; //
         NOLINT
               template<> struct ConwayPolynomial<409, 9> { using ZPZ = aerobus::zpz<409>; using type =
05323
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<818>, ZPZV<318>, ZPZV<211>, ZPZV<388»;
         }; // NOLINT
  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 =
05325
         POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
05326
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<417»; }; // NOLINT
               template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type =
05327
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<373>, ZPZV<2»; }; // NOLINT
05328
               template<> struct ConwayPolynomial<419, 5> { using ZPZ = aerobus::zpz<419>; using type =
        POLYV-ZPZV-1>, ZPZV-(>, ZPZV-(>), ZPZV-(>), ZPZV-(>), ZPZV-(4), ZPZV-417s; }; // NOLINT template<> struct ConwayPolynomial<419, 6> { using ZPZ = aerobus::zpz<419>; using type =
05329
         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<4>, ZPZV<4+, ZPZV<417»; };
05331
              template<> struct ConwayPolynomial<419, 8> { using ZPZ = aerobus::zpz<419>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<234>, ZPZV<388>, ZPZV<151>, ZPZV<2»; }; //
         NOLINT
05332
               template<> struct ConwayPolynomial<419, 9> { using ZPZ = aerobus::zpz<419>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3, ZPZV<3
         }; // NOLINT
05333
               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 =
05334
         POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419»; }; // NOLINT template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
05336
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<421, 5> { using ZPZ = aerobus::zpz<421>; using type =
05337
         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 =
         POLYY<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 =
05339
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<419»; }; // NOLINT template<> struct ConwayPolynomial<421, 8> { using ZPZ = aerobus::zpz<421>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<389>, ZPZV<32>, ZPZV<77>, ZPZV<2»; };
05340
```

```
NOLINT
        template<> struct ConwayPolynomial<421, 9> { using ZPZ = aerobus::zpz<421>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<394>, ZPZV<145>, ZPZV<419»;</pre>
05341
         }; // NOLINT
05342
               template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
        POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
               template<> struct ConwayPolynomial<431, 2> { using ZPZ = aerobus::zpz<431>; using type =
        POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7»; }; // NOLINT
              template<> struct ConwayPolynomial<431, 3> { using ZPZ = aerobus::zpz<431>; using type =
05344
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424»; }; // NOLINT template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
05345
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<431, 5> { using ZPZ = aerobus::zpz<431>; using type =
05346
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<424»; }; // NOLINT
05347
              template<> struct ConwayPolynomial<431, 6> { using ZPZ = aerobus::zpz<431>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<161>, ZPZV<202>, ZPZV<182>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<431, 7> { using ZPZ = aerobus::zpz<431>; using type =
05348
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<243>, ZPZV<286>, ZPZV<115>, ZPZV<7»; }; //
05350
              template<> struct ConwayPolynomial<431, 9> { using ZPZ = aerobus::zpz<431>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<71>, ZPZV<329>, ZPZV<424%;
         }; // NOLINT
05351
               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 =
05352
        POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5»; }; // NOLINT
05353
              template<> struct ConwayPolynomial<433, 3> { using ZPZ = aerobus::zpz<433>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428»; }; // NOLINT template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
05354
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<402>, ZPZV<5»; };
                                                                                             // NOLINT
              template<> struct ConwayPolynomial<433, 5> { using ZPZ = aerobus::zpz<433>; using type =
05355
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<428»; }; // NOLINT
05356
              template<> struct ConwayPolynomial<433, 6> { using ZPZ = aerobus::zpz<433>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<244>, ZPZV<353>, ZPZV<360>, ZPZV<5>; }; // NOLINT template<> struct ConwayPolynomial<433, 7> { using ZPZ = aerobus::zpz<433>; using type
05357
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<428»; }; //
              template<> struct ConwayPolynomial<433, 8> { using ZPZ = aerobus::zpz<433>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347>, ZPZV<32>, ZPZV<39>, ZPZV<5»; };
         NOLINT
05359
        template<> struct ConwayPolynomial<433, 9> { using ZPZ = aerobus::zpz<433>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<27>, ZPZV<232>, ZPZV<45>, ZPZV<428»;</pre>
         }; // NOLINT
              template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
05361
              template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; }; // NOLINT
              template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
05362
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424»; }; // NOLINT
               template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
05363
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; }; // NOLINT
05364
              template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>; }; // NOLINT template<> struct ConwayPolynomial<439, 6> { using ZPZ = aerobus::zpz<439>; using type =
05365
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<324>, ZPZV<190>, ZPZV<15»; }; // NOLINT
               template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; };
              template<> struct ConwayPolynomial<439, 8> { using ZPZ = aerobus::zpz<439>; using type =
         POLYV<ZPZV<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<342>, ZPZV<342>, ZPZV<254>, ZPZV<424»;
05368
         }; // NOLINT
              template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
05369
        POLYV<ZPZV<1>, ZPZV<441»; }; // NOLINT
05370
              template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
        POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
05372
        \label{eq:polyv} \mbox{PDLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2*; }; // \mbox{NOLINT}
              template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<443>; using type =
05373
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44); }; // NOLINT
              template<> struct ConwayPolynomial<443, 6> { using ZPZ = aerobus::zpz<443>; using type =
05374
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<298>, ZPZV<218>, ZPZV<41>, ZPZV<2»; }; // NOLINI
              template<> struct ConwayPolynomial<443, 7> { using ZPZ = aerobus::zpz<443>; using type
05375
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6441»; }; // NOLINT template<> struct ConwayPolynomial<443, 8> { using ZPZ = aerobus::zpz<443>; using type =
05376
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<217>, ZPZV<290>, ZPZV<2»; }; //
         NOLINT
05377
              template<> struct ConwayPolynomial<443, 9> { using ZPZ = aerobus::zpz<443>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<125>, ZPZV<109, ZPZV<441»;
         }; // NOLINT
05378
              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»; };
               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 =
05381
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<449, 5> { using ZPZ = aerobus::zpz<449>; using type =
05382
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<446»; }; // NOLINT
               template<> struct ConwayPolynomial<449, 6> { using ZPZ = aerobus::zpz<449>; using type =
05383
         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 =
05384
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<446»; }; // NOLINT template<> struct ConwayPolynomial<449, 8> { using ZPZ = aerobus::zpz<449>; using type =
05385
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<348>, ZPZV<124>, ZPZV<33*; }; //
05386
              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<29>, ZPZV<446»; };
         // NOLINT
               template<> struct ConwayPolynomial<457, 1> { using ZPZ = aerobus::zpz<457>; using type =
05387
         POLYV<ZPZV<1>, ZPZV<444»; }; // NOLINT
               template<> struct ConwayPolynomial<457, 2> { using ZPZ = aerobus::zpz<457>; using type =
         POLYV<ZPZV<1>, ZPZV<454>, ZPZV<13»; }; // NOLINT
05389
               template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444*; }; // NOLINT
template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13*; }; // NOLINT
05390
               template<> struct ConwayPolynomial<457, 5> { using ZPZ = aerobus::zpz<457>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<444»; }; // NOLINT
05392
               template<> struct ConwayPolynomial<457, 6> { using ZPZ = aerobus::zpz<457>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<205>, ZPZV<389>, ZPZV<266>, ZPZV<13»; }; // NOLINT
05393
              template<> struct ConwayPolynomial<457, 7> { using ZPZ = aerobus::zpz<457>; using type
        POLYVCZPZV<1>, ZPZV<0>, ZPZV<0
05394
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<365>, ZPZV<296>, ZPZV<412>, ZPZV<13»; }; //
         template<> struct ConwayPolynomial<457, 9> { using ZPZ = aerobus::zpz<457>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<354>, ZPZV<84>, ZPZV<444»;</pre>
05395
         }; // NOLINT
               template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
         POLYV<ZPZV<1>, ZPZV<459»; }; // NOLINT
              template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
05397
        POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<461, 3> { using ZPZ = aerobus::zpz<461>; using type =
05398
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459»; }; // NOLINT template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
05399
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2»; }; // NOLINT
05400
               template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<459»; }; // NOLINT
05401
              template<> struct ConwayPolynomial<461, 6> { using ZPZ = aerobus::zpz<461>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<439>, ZPZV<432>, ZPZV<32>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<461, 7> { using ZPZ = aerobus::zpz<461>; using type =
05402
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<459»; };
               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»; }; //
              template<> struct ConwayPolynomial<461, 9> { using ZPZ = aerobus::zpz<461>; using type =
05404
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<210>, ZPZV<276>, ZPZV<459»;
05405
              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 =
05406
        POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
05407
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<460»; }; // NOLINT template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; }; // NOLINT
05409
              template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
         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
05410
05411
               template<> struct ConwayPolynomial<463, 7> { using ZPZ = aerobus::zpz<463>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<460»; }; // NOLI template<> struct ConwayPolynomial<463, 8> { using ZPZ = aerobus::zpz<463>; using type =
05412
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<414>, ZPZV<396>, ZPZV<3»; }; //
         NOLINT
              template<> struct ConwayPolynomial<463, 9> { using ZPZ = aerobus::zpz<463>; using type =
05413
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<433>, ZPZV<4227>, ZPZV<460»;
         }; // NOLINT
05414
               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 =
05415
         POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
05417
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<467, 5> { using ZPZ = aerobus::zpz<467>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
```

```
05419
            template<> struct ConwayPolynomial<467, 6> { using ZPZ = aerobus::zpz<467>; using type =
       POLYY<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<0>, ZPZV<7>, ZPZV<65»; }; // NOLINT template<> struct ConwayPolynomial<467, 8> { using ZPZ = aerobus::zpz<467>; using type =
05421
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<318>, ZPZV<413>, ZPZV<289>, ZPZV<2»; }; //
       NOLINT
            template<> struct ConwayPolynomial<467, 9> { using ZPZ = aerobus::zpz<467>; using type
05422
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39, ZPZV<397>, ZPZV<447>, ZPZV<465»;
       }; // NOLINT
05423
            template<> struct ConwayPolynomial<479, 1> { using ZPZ = aerobus::zpz<479>; using type =
       POLYV<ZPZV<1>, ZPZV<466»; }; // NOLINT
            template<> struct ConwayPolynomial<479, 2> { using ZPZ = aerobus::zpz<479>; using type =
05424
       POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; }; // NOLINT
05425
            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 =
05426
       POLYY<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
            template<> struct ConwayPolynomial<479, 6> { using ZPZ = aerobus::zpz<479>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<243>, ZPZV<287>, ZPZV<334>, ZPZV<13»; }; // NOLINT
05429
           template<> struct ConwayPolynomial<479, 7> { using ZPZ = aerobus::zpz<479>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<466%; }; // NOLINT template<> struct ConwayPolynomial<479, 8> { using ZPZ = aerobus::zpz<479>; using type =
05430
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<247>, ZPZV<440>, ZPZV<17>, ZPZV<13»; }; //
       template<> struct ConwayPolynomial<479, 9> { using ZPZ = aerobus::zpz<479>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<185>, ZPZV<466»; };</pre>
05431
       // NOLINT
            template<> struct ConwayPolynomial<487, 1> { using ZPZ = aerobus::zpz<487>; using type =
05432
       POLYV<ZPZV<1>, ZPZV<484»; }; // NOLINT
            template<> struct ConwayPolynomial<487, 2> { using ZPZ = aerobus::zpz<487>; using type =
05433
       POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; }; // NOLINT
05434
            template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; }; // NOLINT template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3»; }; // NOLINT
05435
            template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
05436
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<484»; }; // NOLINT
05437
            template<> struct ConwayPolynomial<487, 6> { using ZPZ = aerobus::zpz<487>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<45>, ZPZV<427>, ZPZV<185>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<487, 7> { using ZPZ = aerobus::zpz<487>; using type = DOLYVZPZVZ1>, ZPZV<3.
05438
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<484»; };
            template<> struct ConwayPolynomial<487, 8> { using ZPZ = aerobus::zpz<487>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<249>, ZPZV<137>, ZPZV<3»; }; //
       template<> struct ConwayPolynomial<487, 9> { using ZPZ = aerobus::zpz<487>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<271>, ZPZV<4447>, ZPZV<484»;</pre>
05440
       }; // NOLINT
05441
            template<> struct ConwayPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<489»; }; // NOLINT
05442
            template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
05443
       POLYY<ZPZY<1>, ZPZV<0>, ZPZV<2>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
05445
       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 =
05446
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<369>, ZPZV<125>, ZPZV<125>, ZPZV<125>, ZPZV<291>; // NOLINT template<> struct ConwayPolynomial<491, 7> { using ZPZ = aerobus::zpz<491>; using type
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<489»; }; //
05448
            template<> struct ConwayPolynomial<491, 8> { using ZPZ = aerobus::zpz<491>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378>, ZPZV<372>, ZPZV<216>, ZPZV<2); }; //
       NOLINT
05449
           template<> struct ConwayPolynomial<491, 9> { using ZPZ = aerobus::zpz<491>; using type =
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<453>, ZPZV<489»;
       }; // NOLINT
            template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
05450
       POLYV<ZPZV<1>, ZPZV<492»; }; // NOLINT
            template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
05451
       POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
05452
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; }; // NOLINT
            template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
05453
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
05454
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<492»; }; // NOLINT
       template<> struct ConwayPolynomial<499, 6> { using ZPZ = aerobus::zpz<499>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<407>, ZPZV<191>, ZPZV<78>, ZPZV<7»; }; // NOLINT
05455
            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<8>, ZPZV<492»; };
05457
           template<> struct ConwayPolynomial<499, 8> { using ZPZ = aerobus::zpz<499>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<309>, ZPZV<200>, ZPZV<7»; }; //
       NOLTNT
```

```
template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type :
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<491>, ZPZV<222>, ZPZV<492»;
        }; // NOLINT
05459
              template<> struct ConwayPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
        POLYV<ZPZV<1>, ZPZV<498»; }; // NOLINT
              template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
05460
        POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
05461
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<498»; }; // NOLINT template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type =
05462
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<325>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<503, 5> { using ZPZ = aerobus::zpz<503>; using type =
05463
        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 =
        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 =
05465
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<498»; }; // NOLINT
              template<> struct ConwayPolynomial<503, 8> { using ZPZ = aerobus::zpz<503>; using type =
05466
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<441>, ZPZV<203>, ZPZV<316>, ZPZV<5»; }; //
        template<> struct ConwayPolynomial<503, 9> { using ZPZ = aerobus::zpz<503>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<158>, ZPZV<337>, ZPZV<498»;
05467
        }; // NOLINT
               template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
05468
        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
05470
              template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
05471
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<40>, ZPZV<408>, ZPZV<208; }; // NOLINT template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
05472
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
05473
             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 =
05474
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<507»; }; // NOLINT
              template<> struct ConwayPolynomial<509, 8> { using ZPZ = aerobus::zpz<509>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<473>, ZPZV<382>, ZPZV<2»; };
05476
             template<> struct ConwayPolynomial<509, 9> { using ZPZ = aerobus::zpz<509>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3, ZPZV<3,
        }; // NOLINT
05477
               template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
        POLYV<ZPZV<1>, ZPZV<518»; }; // NOLINT
05478
               template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
        POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; }; // NOLINT
05479
              template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518»; }; // NOLINT
              template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
05480
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<50>, ZPZV<509>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<521, 5> { using ZPZ = aerobus::zpz<521>; using type =
05481
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<518»; }; // NOLINT
        template<> struct ConwayPolynomial<521, 6> { using ZPZ = aerobus::zpz<521>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<315>, ZPZV<153>, ZPZV<280>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<521, 7> { using ZPZ = aerobus::zpz<521>; using type =
05482
05483
        POLYV<2PZV<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 =
05484
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<46>, ZPZV<407>, ZPZV<312>, ZPZV<31; //
        NOLINT
              template<> struct ConwayPolynomial<521, 9> { using ZPZ = aerobus::zpz<521>; using type =
05485
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<181>, ZPZV<483>, ZPZV<518»;
        }; // NOLINT
               template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
        POLYV<ZPZV<1>, ZPZV<521»; }; // NOLINT
05487
              template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
        POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial523, 3> { using ZPZ = aerobus::zpz<523>; using type =
05488
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<521»; }; // NOLINT
               template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
05489
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
05490
        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 =
05491
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<475>, ZPZV<371>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<523, 7> { using ZPZ = aerobus::zpz<523>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<521»; };
05493
              template<> struct ConwayPolynomial<523, 8> { using ZPZ = aerobus::zpz<523>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<518>, ZPZV<184>, ZPZV<380>, ZPZV<2»; }; //
        NOLINT
              template<> struct ConwayPolynomial<523, 9> { using ZPZ = aerobus::zpz<523>; using type =
05494
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<342>, ZPZV<345>, ZPZV<521»;
        }; // NOLINT
05495
              template<> struct ConwayPolynomial<541, 1> { using ZPZ = aerobus::zpz<541>; using type =
        POLYY<ZPZV<1>, ZPZV<539»; }; // NOLINT template<> struct ConwayPolynomial<541, 2> { using ZPZ = aerobus::zpz<541>; using type =
05496
        POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; }; // NOLINT
```

```
05497
               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 =
05498
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
05499
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<33>, ZPZV<539»; }; // NOLINT
              template<> struct ConwayPolynomial<541, 6> { using ZPZ = aerobus::zpz<541>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<239>, ZPZV<320>, ZPZV<69>, ZPZV<2»; }; // NOLINI
              template<> struct ConwayPolynomial<541, 7> { using ZPZ = aerobus::zpz<541>; using type =
05501
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<539»; }; // NOLINT
05502
              template<> struct ConwayPolynomial<541, 8> { using ZPZ = aerobus::zpz<541>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<376>, ZPZV<108>, ZPZV<113>, ZPZV<2*; }; //
              template<> struct ConwayPolynomial<541, 9> { using ZPZ = aerobus::zpz<541>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<340>, ZPZV<318>, ZPZV<539»;
         }; // NOLINT
05504
              template<> struct ConwayPolynomial<547, 1> { using ZPZ = aerobus::zpz<547>; using type =
        POLYV<ZPZV<1>, ZPZV<545»; }; // NOLINT
              template<> struct ConwayPolynomial<547, 2> { using ZPZ = aerobus::zpz<547>; using type =
        POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<547, 3> { using ZPZ = aerobus::zpz<547>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 4> { using ZPZ = aerobus::zpz<547>; using type =
05507
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<334>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<547, 5> { using ZPZ = aerobus::zpz<547>; using type =
05508
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<545»; }; // NOLINT
              template<> struct ConwayPolynomial<547, 6> { using ZPZ = aerobus::zpz<547>; using type =
05509
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<547, 7> { using ZPZ = aerobus::zpz<547>; using type
05510
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<545»; }; // NOLINT
              template<> struct ConwayPolynomial<547, 8> { using ZPZ = aerobus::zpz<547>; using type =
05511
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2»; }; //
             template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type =
05512
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<238>, ZPZV<263>, ZPZV<545»;
         }; // NOLINT
        template<> struct ConwayPolynomial<557, 1> { using ZPZ = aerobus::zpz<557>; using type = POLYV<ZPZV<1>, ZPZV<555»; }; // NOLINT
05513
               template<> struct ConwayPolynomial<557, 2> { using ZPZ = aerobus::zpz<557>; using type =
         POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; }; // NOLINT
05515
              template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<555»; }; // NOLINT template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; }; // NOLINT
05516
              template<> struct ConwayPolynomial<557, 5> { using ZPZ = aerobus::zpz<557>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<555»; }; // NOLINT
05518
              template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type =
         \texttt{POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<20>, ZPZV<192>, ZPZV<253>, ZPZV<253>, ZPZV<29; }; \ // \ \texttt{NOLINT}  
              template<> struct ConwayPolynomial<557, 7> { using ZPZ = aerobus::zpz<557>; using type =
05519
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<555»; }; // NOLINT
              template<> struct ConwayPolynomial<557, 8> { using ZPZ = aerobus::zpz<557>; using type
05520
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<384>, ZPZV<113>, ZPZV<2»; }; //
         NOLINT
        \label{eq:convergence} template<> struct ConwayPolynomial<557, 9> \{ using ZPZ = aerobus::zpz<557>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<456>, ZPZV<456>, ZPZV<434>, ZPZV<555»; zPZV<555»; zPZV<555»; zPZV<555»; zPZV<555»; zPZV<5550, ZPZV<550, ZPZV<5550, ZPZV<550, ZPZ
05521
         }; // NOLINT
               template<> struct ConwayPolynomial<563, 1> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<561»; }; // NOLINT
               template<> struct ConwayPolynomial<563, 2> { using ZPZ = aerobus::zpz<563>; using type =
05523
        POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
05524
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; }; // NOLINT template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
05525
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<20>, ZPZV<399>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<563, 5> { using ZPZ = aerobus::zpz<563>; using type =
05526
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<561»; }; // NOLINT
05527
              template<> struct ConwayPolynomial<563, 6> { using ZPZ = aerobus::zpz<563>; using type =
        POLYY-ZPZV-1>, ZPZV-(>), ZPZV-1>, ZPZV-12>, ZPZV-33>, ZPZV-246>, ZPZV-2»; }; // NOLINT template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz-563>; using type
05528
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<561»; };
              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
05530
              template<> struct ConwayPolynomial<563, 9> { using ZPZ = aerobus::zpz<563>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<15>, ZPZV<19>, ZPZV<561»; };
05531
              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 =
05532
        POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
05533
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT
              template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; }; // NOLINT
05535
             template<> struct ConwayPolynomial<569, 5> { using ZPZ = aerobus::zpz<569>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT
05536
              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»; };
                   template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<566»; }; // NOLINT
05538
                 template<> struct ConwayPolynomial<569, 8> { using ZPZ = aerobus::zpz<569>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<527>, ZPZV<173>, ZPZV<241>, ZPZV<3*; }; //
           NOT.TNT
                   template<> struct ConwayPolynomial<569, 9> { using ZPZ = aerobus::zpz<569>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<566>, ZPZV<5660>, ZPZV<566>, ZPZV<566>, ZPZV<566>, ZPZV<566>, ZPZV<566>, ZPZV<
           }; // NOLINT
05540
                   template<> struct ConwayPolynomial<571, 1> { using ZPZ = aerobus::zpz<571>; using type =
           POLYV<ZPZV<1>, ZPZV<568»; }; // NOLINT
                  \texttt{template<>} \texttt{struct ConwayPolynomial<571, 2> \{ \texttt{using ZPZ = aerobus::zpz<571>; using type = 2.5 type = 2
05541
           POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial571, 3> { using ZPZ = aerobus::zpz<571>; using type =
05542
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type =
05543
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<571, 5> { using ZPZ = aerobus::zpz<571>; using type =
05544
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<568»; }; // NOLINT
                   template<> struct ConwayPolynomial<571, 6> { using ZPZ = aerobus::zpz<571>; using type =
           POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<221>, ZPZV<295>, ZPZV<3>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<571, 7> { using ZPZ = aerobus::zpz<571>; using type =
05546
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 8> { using ZPZ = aerobus::zpz<571>; using type =
05547
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<363>, ZPZV<119>, ZPZV<371>, ZPZV<37»; }; //
                  template<> struct ConwayPolynomial<571, 9> { using ZPZ = aerobus::zpz<571>; using type =
05548
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<545>, ZPZV<179>, ZPZV<568»;
           }; // NOLINT
05549
                  template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
           POLYV<ZPZV<1>, ZPZV<572»; }; // NOLINT
05550
                   template<> struct ConwayPolynomial<577, 2> { using ZPZ = aerobus::zpz<577>; using type =
           POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5»; }; // NOLINT
05551
                  template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572»; }; // NOLINT
template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<577, 5> { using ZPZ = aerobus::zpz<577>; using type =
05552
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<572»; }; // NOLINT
          template<> struct ConwayPolynomial<577, 6> { using ZPZ = aerobus::zpz<577>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<450>, ZPZV<25>, ZPZV<283>, ZPZV<5»; }; // NOLINT
05554
           template<> struct ConwayPolynomial<577, 7> { using ZPZ = aerobus::zpz<577>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<572»; }; // NOLINT
05555
                  template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type =
05556
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<3»; }; //
           NOLINT
05557
                  template<> struct ConwayPolynomial<577, 9> { using ZPZ = aerobus::zpz<577>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<576>, ZPZV<449>, ZPZV<572»;
           }; // NOLINT
05558
                   template<> struct ConwavPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
           POLYV<ZPZV<1>, ZPZV<585»; }; // NOLINT
                   template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
           POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<587, 3> { using ZPZ = aerobus::zpz<587>; using type =
05560
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585»; }; // NOLINT template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
05561
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<444>, ZPZV<2»; };
                                                                                                                        // NOLINT
                  template<> struct ConwayPolynomial<587, 5> { using ZPZ = aerobus::zpz<587>; using type =
05562
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<585»; }; // NOLINT
05563
                   template<> struct ConwayPolynomial<587, 6> { using ZPZ = aerobus::zpz<587>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<204>, ZPZV<121>, ZPZV<226>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type
05564
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<585»; }; // NOLINT
                   template<> struct ConwayPolynomial<587, 8> { using ZPZ = aerobus::zpz<587>; using type
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<492>, ZPZV<44>, ZPZV<91>, ZPŽV<2»; };
           NOLINT
05566
           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<55>, ZPZV<585»;</pre>
           }; // NOLINT
05567
                   template<> struct ConwayPolynomial<593, 1> { using ZPZ = aerobus::zpz<593>; using type =
           POLYV<ZPZV<1>, ZPZV<590»; }; // NOLINT
05568
                  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 =
05569
           POLYY<ZPZY<1>, ZPZV<0>, ZPZV<6>, ZPZV<590»; }; // NOLINT template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3»; }; // NOLINT
05571
                   template<> struct ConwayPolynomial<593, 5> { using ZPZ = aerobus::zpz<593>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<590»; }; // NOLINT
05572
                   template<> struct ConwayPolynomial<593, 6> { using ZPZ = aerobus::zpz<593>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<345>, ZPZV<65>, ZPZV<478>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<593,
                                                                                              7> { using ZPZ = aerobus::zpz<593>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<590»; }; // NOL template<> struct ConwayPolynomial<593, 8> { using ZPZ = aerobus::zpz<593>; using type :
05574
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<350>, ZPZV<291>, ZPZV<495>, ZPZV<495>, ZPZV<3»; }; //
           NOLINT
05575
                  template<> struct ConwayPolynomial<593, 9> { using ZPZ = aerobus::zpz<593>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223>, ZPZV<523>, ZPZV<590»;
         }; // NOLINT
05576
              template<> struct ConwayPolynomial<599, 1> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<592»; }; // NOLINT
05577
               template<> struct ConwayPolynomial<599, 2> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
               template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<592»; }; // NOLINT template<> struct ConwayPolynomial<599, 4> { using ZPZ = aerobus::zpz<599>; using type =
05579
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7»; }; // NOLINT
              template<> struct ConwayPolynomial<599, 5> { using ZPZ = aerobus::zpz<599>; using type =
05580
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<592»; }; // NOLINT
              template<> struct ConwayPolynomial<599, 6> { using ZPZ = aerobus::zpz<599>; using type =
05581
         POLYV<2PZV<1>, 2PZV<0>, 2PZV<1>, 2PZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7»; }; // NOLIN
05582
              template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<592»; }; // NOLIN template<> struct ConwayPolynomial<599, 8> { using ZPZ = aerobus::zpz<599>; using type =
05583
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<440>, ZPZV<37>, ZPZV<124>, ZPZV<7»; };
              template<> struct ConwayPolynomial<599, 9> { using ZPZ = aerobus::zpz<599>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<114>, ZPZV<98>, ZPZV<592»;
         }; // NOLINT
05585
              template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type =
        POLYV<ZPZV<1>, ZPZV<594»; }; // NOLINT
05586
               template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
         POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
              template<> struct ConwayPolynomial<601, 3> { using ZPZ = aerobus::zpz<601>; using type =
05587
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<594»; }; // NOLINT template<> struct ConwayPolynomial<601, 4> { using ZPZ = aerobus::zpz<601>; using type =
05588
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<601, 5> { using ZPZ = aerobus::zpz<601>; using type =
05589
        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 =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<440>, ZPZV<49>, ZPZV<7»; }; // NOLINT
05591
              template<> struct ConwayPolynomial<601, 7> { using ZPZ = aerobus::zpz<601>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<594»; }; // NOLINT
              template<> struct ConwayPolynomial<601, 8> { using ZPZ = aerobus::zpz<601>; using type =
05592
         POLYV<ZPZV<1>, ZPZV<0>, ZPŽV<0>, ZPZV<0>, ZPZV<0>, ZPZV<550>, ZPZV<241>, ZPZV<490>, ZPŽV<7»; }; //
              template<> struct ConwayPolynomial<601, 9> { using ZPZ = aerobus::zpz<601>; using type =
05593
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<487>, ZPZV<487>, ZPZV<599>, ZPZV<594»;
         }; // NOLINT
               template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
05594
        POLYV<ZPZV<1>, ZPZV<604»; }; // NOLINT
               template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
        POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; }; // NOLINT
05596
              template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<604»; }; // NOLINT template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
05597
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<449>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<607, 5> { using ZPZ = aerobus::zpz<607>; using type =
05598
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<604»; }; // NOLINT
05599
              template<> struct ConwayPolynomial<607, 6> { using ZPZ = aerobus::zpz<607>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<40>, ZPZV<40>, ZPZV<45>, ZPZV<478>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type =
05600
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<00, ZPZV<00, ZPZV<00, ZPZV<0>, ZPZV<00, ZPZV<00, ZPZV<00, ZPZV<0>, ZPZV<00, ZPZV<0
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<468>, ZPZV<35>, ZPZV<449>, ZPZV<3»; };
05602
               template<> struct ConwayPolynomial<607, 9> { using ZPZ = aerobus::zpz<607>; using type :
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<444>, ZPZV<429>, ZPZV<604»;
         }; // NOLINT
05603
               template<> struct ConwayPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
        POLYV<ZPZV<1>, ZPZV<611»; }; // NOLINT
05604
              template<> struct ConwayPolynomial<613, 2> { using ZPZ = aerobus::zpz<613>; using type =
        POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; }; // NOLINT
05605
              template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<611»; }; // NOLINT template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
05606
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2»; };
                                                                                              // NOLINT
              template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
05607
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<611»; }; // NOLINT
05608
              template<> struct ConwayPolynomial<613, 6> { using ZPZ = aerobus::zpz<613>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<609>, ZPZV<595>, ZPZV<601>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type
05609
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6+, ZPZV<611*, }; //
              template<> struct ConwayPolynomial<613, 8> { using ZPZ = aerobus::zpz<613>; using type
05610
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<489>, ZPZV<57>, ZPZV<539>, ZPZV<2»; };
         NOLINT
        template<> struct ConwayPolynomial<613, 9> { using ZPZ = aerobus::zpz<613>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<513>, ZPZV<516>, ZPZV<611»;</pre>
05611
        }; // NOLINT
    template<> struct ConwayPolynomial<617, 1> { using ZPZ = aerobus::zpz<617>; using type =
        POLYV<ZPZV<1>, ZPZV<614»; }; // NOLINT
              template<> struct ConwayPolynomial<617, 2> { using ZPZ = aerobus::zpz<617>; using type =
05613
        POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
05614
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<614»; };
                                                               // NOLINT
      template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<617, 5> { using ZPZ = aerobus::zpz<617>; using type =
05616
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<614»; }; // NOLINT
      template<> struct ConwayPolynomial<617, 6> { using ZPZ = aerobus::zpz<617>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<318>, ZPZV<595>, ZPZV<310>, ZPZV<3»; }; // NOLINT
05617
            template<> struct ConwayPolynomial<617, 7> { using ZPZ = aerobus::zpz<617>; using type
05618
      POLYY<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 =
05619
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<519>, ZPZV<501>, ZPZV<155>, ZPZV<3»; }; //
       NOLINT
           template<> struct ConwayPolynomial<617, 9> { using ZPZ = aerobus::zpz<617>; using type =
05620
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<543>, ZPZV<614»;
       }; // NOLINT
05621
           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 =
05622
       POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
05623
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<6>, ZPZY<617»; }; // NOLINT template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
05624
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<619, 5> { using ZPZ = aerobus::zpz<619>; using type =
05625
      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>, 2PZV<0>, 2PZV<238>, ZPZV<468>, ZPZV<347>, ZPZV<2»; }; // NOLINI
05627
           template<> struct ConwayPolynomial<619, 7> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<617»; }; // NOLINT
05628
           template<> struct ConwayPolynomial<619, 8> { using ZPZ = aerobus::zpz<619>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<416>, ZPZV<383>, ZPZV<225>, ZPZV<2*; }; //
       NOLINT
           template<> struct ConwayPolynomial<619, 9> { using ZPZ = aerobus::zpz<619>; using type =
05629
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<510>, ZPZV<617»;
       }; // NOLINT
05630
            template<> struct ConwayPolynomial<631, 1> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<628»; }; // NOLINT
            template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<631, 3> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT template<> struct ConwayPolynomial<631, 4> { using ZPZ = aerobus::zpz<631>; using type =
05633
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<3%; }; // NOLINT template<> struct ConwayPolynomial<631, 5> { using ZPZ = aerobus::zpz<631>; using type =
05634
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
05635
           template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
       \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 516>, \ \texttt{ZPZV} < 541>, \ \texttt{ZPZV} < 106>, \ \texttt{ZPZV} < 3 \text{*; } \ // \ \texttt{NOLINT} 
05636
           template<> struct ConwayPolynomial<631, 7> { using ZPZ = aerobus::zpz<631>; 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>; Using ZPZ = aerobus::zpz<631>; using type =
05637
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<3»; }; //
05638
           template<> struct ConwayPolynomial<631, 9> { using ZPZ = aerobus::zpz<631>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<296>, ZPZV<413>, ZPZV<628»;
       }; // NOLINT
05639
            template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<638»; }; // NOLINT
           template<> struct ConwayPolynomial<641, 2> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3»; }; // NOLINT
05641
            template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<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
05642
            template<> struct ConwayPolynomial<641, 5> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<638»; }; // NOLINT
05644
           template<> struct ConwayPolynomial<641, 6> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<557>, ZPZV<294>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type =
05645
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<638»; }; // NOLINT
            template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>; using type
05646
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<33»; }; //
      template<> struct ConwayPolynomial<641, 9> { using ZPZ = aerobus::zpz<641>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<141>, ZPZV<638»;</pre>
05647
       }; // NOLINT
           template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<632»; }; // NOLINT
05649
           template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
05650
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<632»; }; // NOLINT
            template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
05651
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<600>, ZPZV<11»; }; // NOLINT
05652
           template<> struct ConwayPolynomial<643, 5> { using ZPZ = aerobus::zpz<643>; using type =
      template<> struct ConwayPolynomial<643, 6> { using ZPZ = aerobus::zpz<643>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<345>, ZPZV<412>, ZPZV<293>, ZPZV<11»; }; // NOLINT</pre>
05653
```

```
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<632»; }; // NOLINT
             template<> struct ConwayPolynomial<643, 8> { using ZPZ = aerobus::zpz<643>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<631>, ZPZV<573>, ZPZV<569>, ZPZV<11»; }; //
        NOLINT
             template<> struct ConwayPolynomial<643, 9> { using ZPZ = aerobus::zpz<643>; using type =
05656
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<591>, ZPZV<475>, ZPZV<632»;
        }; // NOLINT
              template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
05657
        POLYV<ZPZV<1>, ZPZV<642»; }; // NOLINT
              template<> struct ConwayPolynomial<647, 2> { using ZPZ = aerobus::zpz<647>; using type =
05658
        POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<647, 3> { using ZPZ = aerobus::zpz<647>; using type =
05659
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<642»; }; // NOLINT
05660
             template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5»; ); // NOLINT template<> struct ConwayPolynomial<647, 5> { using ZPZ = aerobus::zpz<647>; using type =
05661
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0, ZPZV<1>, ZPZV<642»; }; // NOLINT template<> struct ConwayPolynomial<647, 6> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<308>, ZPZV<385>, ZPZV<642>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<647, 7> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, 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 =
05664
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<603>, ZPZV<259>, ZPZV<271>, ZPZV<5»; }; //
        NOLINT
              template<> struct ConwayPolynomial<647, 9> { using ZPZ = aerobus::zpz<647>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<561>, ZPZV<123>, ZPZV<642»;
        }; // NOLINT
05666
              template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<651»; }; // NOLINT
             template<> struct ConwayPolynomial<653, 2> { using ZPZ = aerobus::zpz<653>; using type =
05667
        POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2»; }; // NOLINT
              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 =
05669
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2»; }; // NOLINT
             template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
05670
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651»; }; // NOLINT
05671
              template<> struct ConwayPolynomial<653, 6> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<45>, ZPZV<220>, ZPZV<242>, ZPZV<24>; }; // NOLINT
05672
             template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZPZV<5, ZPZV<651»; }; // NOLINT template<> struct ConwayPolynomial<653, 8> { using ZPZ = aerobus::zpz<653>; using type =
05673
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<385>, ZPZV<18>, ZPZV<296>, ZPZV<2»; }; //
05674
              template<> struct ConwayPolynomial<653, 9> { using ZPZ = aerobus::zpz<653>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<651»;
        }; // NOLINT
05675
              template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<657»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2»; }; // NOLINT
05677
              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 =
05678
        POLYV<ZPZV<1>, ZPZV<6>, ZPZV<8>, ZPZV<351>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<657»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 6> { using ZPZ = aerobus::zpz<659>; using type =
05680
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<371>, ZPZV<105>, ZPZV<223>, ZPZV<2»; }; // NOLINT
05681
             template<> struct ConwayPolynomial<659, 7> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<657»; }; // NOLINT
              template<> struct ConwayPolynomial<659, 8> { using ZPZ = aerobus::zpz<659>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<358>, ZPZV<246>, ZPZV<90>, ZPZV<2»; };
        template<> struct ConwayPolynomial<659, 9> { using ZPZ = aerobus::zpz<659>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592>, ZPZV<592>, ZPZV<46>, ZPZV<657»;</pre>
05683
        }; // NOLINT
              template<> struct ConwayPolynomial<661, 1> { using ZPZ = aerobus::zpz<661>; using type =
05684
        POLYV<ZPZV<1>, ZPZV<659»; }; // NOLINT
              template<> struct ConwayPolynomial<661, 2> { using ZPZ = aerobus::zpz<661>; using type =
05685
        POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
05686
        POLYV<ZPZV<1>, ZPZV<0, ZPZV<4>, ZPZV<659»; }; // NOLINT template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2»; }; // NOLINT
05687
             template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
05688
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<659»; }; // NOLINT
05689
              template<> struct ConwayPolynomial<661, 6> { using ZPZ = aerobus::zpz<661>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<456>, ZPZV<382>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<661, 7> { using ZPZ = aerobus::zpz<661>; using type = aerobus::zpz<
05690
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<659»; };
              template<> struct ConwayPolynomial<661, 8> { using ZPZ = aerobus::zpz<661>; using type
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<612>, ZPZV<285>, ZPZV<72>, ZPZV<2»; };
        template<> struct ConwayPolynomial<661, 9> { using ZPZ = aerobus::zpz<661>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<389>, ZPZV<220>, ZPZV<659»;</pre>
05692
```

```
}; // NOLINT
05693
                   template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
           POLYY<ZPZV<1>, ZPZV<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
05694
           POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5»; }; // NOLINT
                   template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
05695
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
05696
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<61>, ZPZV<5>; }; // NOLINT

template<> struct ConwayPolynomial<673, 5> { using ZPZ = aerobus::zpz<673>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<68*; }; // NOLINT

template<> struct ConwayPolynomial<673, 6> { using ZPZ = aerobus::zpz<673>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<0 , ZP
05697
05698
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<524>, ZPZV<248>, ZPZV<35>, ZPZV<5»; };
                   template<> struct ConwayPolynomial<673, 7> { using ZPZ = aerobus::zpz<673>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66, ZPZV<668»; }; // NOLINT
05700
                  template<> struct ConwayPolynomial<673, 8> { using ZPZ = aerobus::zpz<673>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<669>, ZPZV<587>, ZPZV<302>, ZPZV<5»; }; //
           NOLINT
                   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
05702
                   template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
           POLYV<ZPZV<1>, ZPZV<675»; }; // NOLINT
                  template<> struct ConwayPolynomial677, 2> { using ZPZ = aerobus::zpz<677>; using type =
05703
           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 =
05705
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<677, 5> { using ZPZ = aerobus::zpz<677>; using type =
05706
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<675»; }; // NOLINT
05707
                   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
05708
                 template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<675»; }; // NOLINT template<> struct ConwayPolynomial<677, 8> { using ZPZ = aerobus::zpz<677>; using type =
05709
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<619>, ZPZV<619>, ZPZV<152>, ZPZV<2»; }; //
05710
                   template<> struct ConwayPolynomial<677, 9> { using ZPZ = aerobus::zpz<677>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<504>, ZPZV<404>, ZPZV<675»;
           }; // NOLINT
0.5711
                   template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
           POLYV<ZPZV<1>, ZPZV<678»; }; // NOLINT
                  template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
           POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; }; // NOLINT
05713
                   template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<678»; }; // NOLINT
template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<683, 5> { using ZPZ = aerobus::zpz<683>; using type =
05714
05715
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<678»; }; // NOLINT
                   template<> struct ConwayPolynomial<683, 6> { using ZPZ = aerobus::zpz<683>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<644>, ZPZV<109>, ZPZV<434>, ZPZV<5»; }; // NOLINT
05717
                  template<> struct ConwayPolynomial<683, 7> { using ZPZ = aerobus::zpz<683>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6, ZPZV<0>, ZPZV<6, ZPZV<6,
05718
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<383>, ZPZV<184>, ZPZV<65>, ZPZV<55»; };
           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»;
05719
           }; // NOLINT
          template<> struct ConwayPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
POLYV<ZPZV<1>, ZPZV<688»; }; // NOLINT</pre>
05720
                   template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
           POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
05722
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<688»; }; // NOLINT
template<> struct ConwayPolynomial<691, 4> { using ZPZ = aerobus::zpz<691>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; }; // NOLINT
05723
                   template<> struct ConwayPolynomial<691, 5> { using ZPZ = aerobus::zpz<691>; using type =
05724
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; }; // NOLINT
05725
                  template<> struct ConwayPolynomial<691, 6> { using ZPZ = aerobus::zpz<691>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<408>, ZPZV<262>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type =
05726
           POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; };
                                                                                                                                                                          // NOLINT
                   template<> struct ConwayPolynomial<691, 8> { using ZPZ = aerobus::zpz<691>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<356>, ZPZV<425>, ZPZV<321>, ZPZV<3»; };
05728
                  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<5>, ZPZV<55>, ZPZV<443>, ZPZV<688»;
           }; // NOLINT
                   template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
           POLYV<ZPZV<1>, ZPZV<699»; }; // NOLINT
05730
                  template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
          POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
05731
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; }; // NOLINT
```

```
05732
            template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699»; }; // NOLINT
      template<> struct ConwayPolynomial<701, 6> { using ZPZ = aerobus::zpz<701>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<571>, ZPZV<327>, ZPZV<285>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<701, 7> { using ZPZ = aerobus::zpz<701>; using type =
05734
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<699»; }; //
           template<> struct ConwayPolynomial<701, 8> { using ZPZ = aerobus::zpz<701>; using type =
05736
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<619>, ZPZV<206>, ZPZV<593>, ZPZV<2»; }; //
      NOLINT
           template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type =
05737
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<459>, ZPZV<373>, ZPZV<699»;
       }; // NOLINT
05738
           template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<707»; }; // NOLINT
           template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
05739
      POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707»; }; // NOLINT template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
05741
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<709, 5> { using ZPZ = aerobus::zpz<709>; using type =
0.5742
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<707»; }; // NOLINT
05743
            template<> struct ConwayPolynomial<709, 6> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<669>, ZPZV<514>, ZPZV<295>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<709, 7> { using ZPZ = aerobus::zpz<709>; using type =
05744
      POLYY<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 =
05745
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<689>, ZPZV<233>, ZPZV<79>, ZPZV<2»; }; //
      NOLINT
05746
           template<> struct ConwayPolynomial<709, 9> { using ZPZ = aerobus::zpz<709>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<257>, ZPZV<171>, ZPZV<707»;
       }; // NOLINT
05747
           template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<708»; }; // NOLINT
           template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
05748
      POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; }; // NOLINT
05749
            template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type =
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZV<708»; }; // NOLINT template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
05750
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<602>, ZPZV<61>; }; // NOLINT
template<> struct ConwayPolynomial<719, 5> { using ZPZ = aerobus::zpz<719>; using type =
05751
      POLYY<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 =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<533>, ZPZV<591>, ZPZV<182>, ZPZV<11»; }; // NOLINT
05753
           template<> struct ConwayPolynomial<719, 7> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
           template<> struct ConwayPolynomial<719, 8> { using ZPZ = aerobus::zpz<719>; using type =
05754
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<714>, ZPZV<362>, ZPZV<244>, ZPZV<11»; }; //
           template<> struct ConwayPolynomial<719, 9> { using ZPZ = aerobus::zpz<719>; using type =
05755
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<288>, ZPZV<560>, ZPZV<708»;
       }; // NOLINT
05756
           template<> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<722»; }; // NOLINT
            template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; }; // NOLINT
            template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
05758
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, VPZV<72; ; // NOLINT
template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5>; // NOLINT
template<> struct ConwayPolynomial<727, 5> { using ZPZ = aerobus::zpz<727>; using type =
05759
05760
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<722»; }; // NOLINT
05761
           template<> struct ConwayPolynomial<727, 6> { using ZPZ = aerobus::zpz<727>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<86>, ZPZV<397>, ZPZV<672>, ZPZV<55; }; // NOLINT template<> struct ConwayPolynomial<727, 7> { using ZPZ = aerobus::zpz<727>; using type =
05762
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<722»; }; // NOLINT
           template<> struct ConwayPolynomial<727, 8> { using ZPZ = aerobus::zpz<727>; using type =
05763
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<63>, ZPZV<639>, ZPZV<671>, ZPZV<368>, ZPZV<5»; }; //
05764
           template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<573>, ZPZV<502>, ZPZV<722»;
       }; // NOLINT
05765
            template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<727»; }; // NOLINT
           template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
05766
      POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; }; // NOLINT
           template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
05767
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727»; ); // NOLINT template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
05768
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<6»; };
                                                                          // NOLINT
           template<> struct ConwayPolynomial<733, 5> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<727»; }; // NOLINT
05770
           template<> struct ConwayPolynomial<733, 6> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<549>, ZPZV<151>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<733, 7> { using ZPZ = aerobus::zpz<733>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<727»; };
              template<> struct ConwayPolynomial<733, 8> { using ZPZ = aerobus::zpz<733>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<532>, ZPZV<610>, ZPZV<142>, ZPZV<6»; }; //
        NOLINT
05773
              template<> struct ConwayPolynomial<733, 9> { using ZPZ = aerobus::zpz<733>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<337>, ZPZV<6>, ZPZV<727»; };
              template<> struct ConwayPolynomial<739, 1> { using ZPZ = aerobus::zpz<739>; using type =
05774
        POLYV<ZPZV<1>, ZPZV<736»; }; // NOLINT
             template<> struct ConwayPolynomial<739, 2> { using ZPZ = aerobus::zpz<739>; using type =
05775
        POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; }; // NOLINT
              05776
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; }; // NOLINT template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<3»; }; // NOLINT
05778
              template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<736»; }; // NOLINT
              template<> struct ConwayPolynomial<739, 6> { using ZPZ = aerobus::zpz<739>; using type =
05779
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<447>, ZPZV<625>, ZPZV<3»; }; // NOLINT
05780
              template<> struct ConwayPolynomial<739,
                                                                      7> { using ZPZ = aerobus::zpz<739>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<44>, ZPZV<736»; }; //
05781
             template<> struct ConwayPolynomial<739, 8> { using ZPZ = aerobus::zpz<739>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<401>, ZPZV<169>, ZPZV<25>, ZPZV<3»; }; //
        NOLINT
05782
              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<81b, ZPZV<736»;
        }; // NOLINT
05783
              template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<738»; }; // NOLINT
             template<> struct ConwayPolynomial<743, 2> { using ZPZ = aerobus::zpz<743>; using type =
05784
        POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; }; // NOLINT
05785
              template<> struct ConwayPolynomial<743, 3> { using ZPZ = aerobus::zpz<743>; using type =
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZV<3*, ZPZV<738*; }; // NOLINT template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
05786
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<743, 5> { using ZPZ = aerobus::zpz<743>; using type =
05787
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT
              template<> struct ConwayPolynomial<743, 6> { using ZPZ = aerobus::zpz<743>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<236>, ZPZV<471>, ZPZV<88>, ZPZV<5»; }; // NOLINT
             template<> struct ConwayPolynomial<743, 7> { using ZPZ = aerobus::zpz<743>; using type =
05789
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
             template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
05790
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<279>, ZPZV<588>, ZPZV<58; }; //
              template<> struct ConwayPolynomial<743, 9> { using ZPZ = aerobus::zpz<743>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<327>, ZPZV<676>, ZPZV<738»;
        }; // NOLINT
05792
              template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
        POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
              template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
05793
        POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
05794
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; }; // NOLINT template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
05795
        POLYV<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<5Z5>, ZPZV<5Z5>, ZPZV<5Z5>, ZPZV<5x; }; // NOLINT template<> struct ConwayPolynomial<751, 5> { using ZPZ = aerobus::zpz<751>; using type =
05796
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT
             template<> struct ConwayPolynomial<751, 6> { using ZPZ = aerobus::zpz<751>; using type =
05797
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<298>, ZPZV<633>, ZPZV<539>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<751, 7> { using ZPZ = aerobus::zpz<751>; using type =
05798
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT template<> struct ConwayPolynomial<751, 8> { using ZPZ = aerobus::zpz<751>; using type =
05799
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<741>, ZPZV<243>, ZPZV<672>, ZPZV<672>, ZPZV<3»; }; //
05800
             template<> struct ConwayPolynomial<751, 9> { using ZPZ = aerobus::zpz<751>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<703>, ZPZV<489>, ZPZV<748»;
        }; // NOLINT
05801
              template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
        POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
05802
              template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
        POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2»; }; // NOLINT
05803
             template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
05804
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<757, 5> { using ZPZ = aerobus::zpz<757>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<755»; }; // NOLINT
05806
              template<> struct ConwayPolynomial<757, 6> { using ZPZ = aerobus::zpz<757>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<753>, ZPZV<739>, ZPZV<745>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<757, 7> { using ZPZ = aerobus::zpz<757>; using type =
05807
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<755»; }; // NOLINT
              template<> struct ConwayPolynomial<757, 8> { using ZPZ = aerobus::zpz<757>; using type =
05808
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<110>, ZPZV<509>, ZPZV<2»; }; //
        NOLINT
05809
             template<> struct ConwayPolynomial<757, 9> { using ZPZ = aerobus::zpz<757>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<68*, ZPZV<688*, ZPZV<608*, ZPZV<702*, ZPZV<755*;
        }; // NOLINT
```

```
05810
           template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
05811
          template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; }; // NOLINT
05812
           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
05814
          template<> struct ConwayPolynomial<761, 5> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<5>, ZPZV<755»; }; // NOLINT
05815
           template<> struct ConwayPolynomial<761, 6> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<634>, ZPZV<597>, ZPZV<155>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<761, 7> { using ZPZ = aerobus::zpz<761>; using type
05816
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<755»; }; //
05817
          template<> struct ConwayPolynomial<761, 8> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<603>, ZPZV<144>, ZPZV<540>, ZPZV<540>; }; //
      NOT.TNT
      template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<317>, ZPZV<571>, ZPZV<755»;
05818
05819
           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 =
05820
      POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
05821
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 4> { using ZPZ = aerobus::zpz<769>; using type =
05822
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<769, 5> { using ZPZ = aerobus::zpz<769>; using type =
05823
      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 =
05824
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<326>, ZPZV<650>, ZPZV<11»; };
                                                                                               // NOLINT
           template<> struct ConwayPolynomial<769, 7> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<758»; }; // NOLINT
          template<> struct ConwayPolynomial<769, 8> { using ZPZ = aerobus::zpz<769>; using type =
05826
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<560>, ZPZV<574>, ZPZV<632>, ZPZV<61»; }; //
      NOLINT
           template<> struct ConwayPolynomial<769, 9> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<751>, ZPZV<758»;
      }; // NOLINT
05828
           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 =
05829
      POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<771»; };
                                                            // NOLINT
          template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
05831
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<44, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<773, 5> { using ZPZ = aerobus::zpz<773>; using type =
05832
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<771»; }; // NOLINT
05833
           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<2»; }; // NOLINT
05834
          template<> struct ConwayPolynomial<773, 7> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<771»; }; // NOLINT template<> struct ConwayPolynomial<773, 8> { using ZPZ = aerobus::zpz<773>; using type =
05835
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<484>, ZPZV<94>, ZPZV<693>, ZPZV<2»; };
05836
          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
05837
           template<> struct ConwayPolynomial<787, 1> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<785»; }; // NOLINT
05838
           template<> struct ConwayPolynomial<787, 2> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
          template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
05839
      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
template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
05840
05841
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<785»; }; // NOLINT
           template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
05842
      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 =
05843
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<785»; }; // NOLINT template<> struct ConwayPolynomial<787, 8> { using ZPZ = aerobus::zpz<787>; using type =
05844
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<612>, ZPZV<26>, ZPZV<715>, ZPZV<22»; };
      NOLINT
05845
          template<> struct ConwayPolynomial<787, 9> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<480>, ZPZV<573>, ZPZV<785»;
      }; // NOLINT
05846
           template<> struct ConwayPolynomial<797, 1> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<795»; };
                                       // NOLINT
           template<> struct ConwayPolynomial<797, 2> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»; }; // NOLINT
          template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
05848
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type =
05849
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<717>, ZPZV<2»; };
            template<> struct ConwayPolynomial<797, 5> { using ZPZ = aerobus::zpz<797>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<795»; }; // NOLINT
            template<> struct ConwayPolynomial<797, 6> { using ZPZ = aerobus::zpz<797>; using type =
05851
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<657>, ZPZV<396>, ZPZV<71>, ZPZV<2»; }; // NOLINT
05852
            template<> struct ConwayPolynomial<797,
                                                            7> { using ZPZ = aerobus::zpz<797>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<795»; }; // NOLINT
            template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>;
05853
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2»; }; //
       template<> struct ConwayPolynomial<797, 9> { using ZPZ = aerobus::zpz<797>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<240>, ZPZV<240>, ZPZV<599>, ZPZV<795»;
05854
       }; // NOLINT
   template<> struct ConwayPolynomial<809, 1> { using ZPZ = aerobus::zpz<809>; using type =
       POLYV<ZPZV<1>, ZPZV<806»; }; // NOLINT
05856
           template<> struct ConwayPolynomial<809, 2> { using ZPZ = aerobus::zpz<809>; using type =
       POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
            template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
05857
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<806»; }; // NOLINT
            template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; }; // NOLINT
05859
            template<> struct ConwayPolynomial<809, 5> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, 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
05860
            template<> struct ConwayPolynomial<809, 7> { using ZPZ = aerobus::zpz<809>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; };
05862
           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<3»; }; //
       NOLINT
           template<> struct ConwayPolynomial<809, 9> { using ZPZ = aerobus::zpz<809>; using type =
05863
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<727>, ZPZV<806»;
       }; // NOLINT
            template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
05864
       POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
            template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
05865
       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 =
05867
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3»; }; // NOLINT
       template<> struct ConwayPolynomial x811, 5> { using ZPZ = aerobus::zpz<811>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<808»; }; // NOLINT
05868
            template<> struct ConwayPolynomial<811, 6> { using ZPZ = aerobus::zpz<811>; using type =
05869
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<780>, ZPZV<755>, ZPZV<307>, ZPZV<3»; }; // NOLINT
05870
            template<> struct ConwayPolynomial<811, 7> { using ZPZ = aerobus::zpz<811>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<808»; }; // NOLINT template<> struct ConwayPolynomial<811, 8> { using ZPZ = aerobus::zpz<811>; using type =
05871
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<3»; }; //
       NOLINT
05872
            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<380>, ZPZV<808»;
       }; // NOLINT
05873
            template<> struct ConwayPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
       POLYV<ZPZV<1>, ZPZV<819»; }; // NOLINT
            template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
05874
       POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<821, 3> { using ZPZ = aerobus::zpz<821>; using type =
05875
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<819»; }; // NOLINT
template<> struct ConwayPolynomial<821, 4> { using ZPZ = aerobus::zpz<821>; using type =
05876
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<662>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
05877
       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 =
       POLYV<2PZV<1>, 2PZV<0>, 2PZV<2>, 2PZV<160>, ZPZV<130>, ZPZV<803>, ZPZV<2»; }; // NOLINT
05879
           template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<819»; }; // NOLINT template<> struct ConwayPolynomial<821, 8> { using ZPZ = aerobus::zpz<821>; using type =
05880
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<55>, ZPZV<556>, ZPZV<589>, ZPZV<589>, ZPZV<2»; }; //
       NOLINT
            template<> struct ConwayPolynomial<821, 9> { using ZPZ = aerobus::zpz<821>; using type =
05881
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<650>, ZPZV<557>, ZPZV<819»;
       }; // NOLINT
05882
            template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
       POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
            template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
       POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; }; // NOLINT
05884
            template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
05885
05886
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
05887
           template<> struct ConwayPolynomial<823, 6> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<82>, ZPZV<616>, ZPZV<744>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<820»; }; // NOLINT
05888
```

```
template<> struct ConwayPolynomial<823, 8> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<451>, ZPZV<437>, ZPZV<31>, ZPZV<3»; };
      NOLINT
      template<> struct ConwayPolynomial<823, 9> { using ZPZ = aerobus::zpz<823>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<740>, ZPZV<609>, ZPZV<820»;</pre>
05890
      }; // NOLINT
05891
           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 =
05892
      POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
05893
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825»; }; // NOLINT
           template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type =
05894
                                                                        // NOLINT
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2»; };
05895
           template<> struct ConwayPolynomial<827, 5> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<82>»; }; // NOLINT template<> struct ConwayPolynomial<827, 6> { using ZPZ = aerobus::zpz<827>; using type =
05896
      POLYYCZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<685>, ZPZV<601>, ZPZV<691>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<827, 7> { using ZPZ = aerobus::zpz<827>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<825»; };
           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<32>; };
      NOLINT
      template<> struct ConwayPolynomial<827, 9> { using ZPZ = aerobus::zpz<827>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<372>, ZPZV<825»;</pre>
05899
      }; // NOLINT
           template<> struct ConwayPolynomial<829, 1> { using ZPZ = aerobus::zpz<829>; using type =
05900
      POLYV<ZPZV<1>, ZPZV<827»; }; // NOLINT
05901
           template<> struct ConwayPolynomial<829, 2> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
05902
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<827»; }; // NOLINT
           template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
05903
      05904
           template<> struct ConwayPolynomial<829, 5> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<827»; }; // NOLINT
      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
05905
05906
           template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<827»; };
05907
          template<> struct ConwayPolynomial<829, 8> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<468>, ZPZV<241>, ZPZV<138>, ZPZV<2»; }; //
      NOLINT
05908
           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
05909
           template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<828»; }; // NOLINT
           template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
05910
      POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; }; // NOLINT
05911
           template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; }; // NOLINT
05912
           template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11»; }; // NOLINT template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
05913
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<828»; }; // NOLINT
           template<> struct ConwayPolynomial<839, 6> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<370>, ZPZV<537>, ZPZV<23>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type =
05915
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<828»; }; // NOLIN template<> struct ConwayPolynomial<839, 8> { using ZPZ = aerobus::zpz<839>; using type =
05916
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<553>, ZPZV<779>, ZPZV<329>, ZPZV<11»; }; //
      NOLINT
           template<> struct ConwayPolynomial<839, 9> { using ZPZ = aerobus::zpz<839>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<349>, ZPZV<206>, ZPZV<828»;
      }; // NOLINT
05918
           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 =
05919
      POLYV<ZPZV<1>, ZPZV<852>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<853, 3> { using ZPZ = aerobus::zpz<853>; using type =
05920
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
05921
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<623>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
05922
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<851»; }; // NOLINT
           template<> struct ConwayPolynomial<853, 6> { using ZPZ = aerobus::zpz<853>; using type =
05923
      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 =
05924
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<851»; }; // NOLINT
           template<> struct ConwayPolynomial<853, 8> { using ZPZ = aerobus::zpz<853>; using type
05925
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8+, ZPZV<844>, ZPZV<846>, ZPZV<118>, ZPZV<2»; }; //
05926
           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<11>, ZPZV<677>, ZPZV<821>, ZPZV<851»;
      }; // NOLINT
05927
           template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<854»; };
               template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
         POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
05929
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; }; // NOLINT template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
05930
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<528>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<857, 5> { using ZPZ = aerobus::zpz<857>; using type =
05931
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<854»; }; // NOLINT
        template<> struct ConwayPolynomial<857, 6> { using ZPZ = aerobus::zpz<857>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<824>, ZPZV<65>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::zpz<857>; using type =
05932
05933
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<854»; };
               template<> struct ConwayPolynomial<857, 8> { using ZPZ = aerobus::zpz<857>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<611>, ZPZV<552>, ZPZV<494>, ZPZV<3»; }; //
              template<> struct ConwayPolynomial<857, 9> { using ZPZ = aerobus::zpz<857>; using type =
05935
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<319>, ZPZV<719>, ZPZV<854»;
         }; // NOLINT
               template<> struct ConwayPolynomial<859, 1> { using ZPZ = aerobus::zpz<859>; using type =
         POLYV<ZPZV<1>, ZPZV<857»; }; // NOLINT
               template<> struct ConwayPolynomial<859, 2> { using ZPZ = aerobus::zpz<859>; using type =
05937
        POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
05938
        POLYV<ZPZV<1>, ZPZV<6>, ZPZV<65, ZPZV<65, ZPZV<65, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2»; }; // NOLINT
05940
               template<> struct ConwayPolynomial<859, 5> { using ZPZ = aerobus::zpz<859>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<857»; }; // NOLINT
        template<> struct ConwayPolynomial<859, 6> { using ZPZ = aerobus::zpz<859>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<419>, ZPZV<646>, ZPZV<566>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpz<859>; using type =
05941
05942
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<857»; }; //
05943
              template<> struct ConwayPolynomial<859, 8> { using ZPZ = aerobus::zpz<859>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<522>, ZPZV<446>, ZPZV<672>, ZPZV<62»; }; //
         NOLINT
         template<> struct ConwayPolynomial<859, 9> { using ZPZ = aerobus::zpz<859>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<648>, ZPZV<845>, ZPZV<857»;
05944
         }; // NOLINT
               template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
05945
         POLYV<ZPZV<1>, ZPZV<858»; }; // NOLINT
               template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
05946
         POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»: }: // NOLINT
               template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
05947
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<858»; }; // NOLINT template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
05948
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<863, 5> { using ZPZ = aerobus::zpz<863>; using type =
05949
         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 =
05950
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<330>, ZPZV<62>, ZPZV<300>, ZPZV<5»; };
               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<858»; }; // NOLINT
05952
              template<> struct ConwayPolynomial<863, 8> { using ZPZ = aerobus::zpz<863>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<765>, ZPZV<576>, ZPZV<849>, ZPZV<5»; }; //
         NOLINT
               template<> struct ConwayPolynomial<863, 9> { using ZPZ = aerobus::zpz<863>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<381>, ZPZV<1>, ZPZV<858»; };
         // NOLINT
05954
               template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
         POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
05955
         POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875»; }; // NOLINT template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type =
05957
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<877, 5> { using ZPZ = aerobus::zpz<877>; using type =
05958
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<875»; }; // NOLINT
05959
               template<> struct ConwayPolynomial<877, 6> { using ZPZ = aerobus::zpz<877>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<400>, ZPZV<855>, ZPZV<2»; }; // NOLINT
05960
              template<> struct ConwayPolynomial<877, 7> { using ZPZ = aerobus::zpz<877>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<875»; }; // NOLINT template<> struct ConwayPolynomial<877, 8> { using ZPZ = aerobus::zpz<877>; using type =
05961
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<767>, ZPZV<319>, ZPZV<347>, ZPZV<34>, ZPZV<34>, ZPZV<319>, ZPZV<347>, ZPZV<347</p>
              template<> struct ConwayPolynomial<877, 9> { using ZPZ = aerobus::zpz<877>; using type
05962
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<770>, ZPZV<78>, ZPZV<875»;
         }; // NOLINT
05963
               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 =
         POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT
05965
              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 = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<447>, ZPZV<3»; }; // NOLINT
05966
```

```
05967
                  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
                 template<> struct ConwayPolynomial<881, 6> { using ZPZ = aerobus::zpz<881>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<419>, ZPZV<211>, ZPZV<231>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<881, 7> { using ZPZ = aerobus::zpz<881>; using type
05969
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<878»; };
                                                                                                                                                                 // NOLINT
                  template<> struct ConwayPolynomial<881, 8> { using ZPZ = aerobus::zpz<881>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<490>, ZPZV<561>, ZPZV<50>; ;;
          template<> struct ConwayPolynomial<881, 9> { using ZPZ = aerobus::zpz<881>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<587>, ZPZV<510>, ZPZV<878»;</pre>
05971
           }; // NOLINT
05972
                  template<> struct ConwayPolynomial<883, 1> { using ZPZ = aerobus::zpz<883>; using type =
          POLYV<ZPZV<1>, ZPZV<881»; }; // NOLINT
05973
                  template<> struct ConwayPolynomial<883, 2> { using ZPZ = aerobus::zpz<883>; using type =
          POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<883, 3> { using ZPZ = aerobus::zpz<883>; using type =
05974
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<881»; };
                                                                                                // NOLINT
                  template<> struct ConwayPolynomial<883, 4> { using ZPZ = aerobus::zpz<883>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<883, 5> { using ZPZ = aerobus::zpz<883>; using type =
05976
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<881»; }; // NOLINT
05977
                  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 template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type
05978
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
05979
                  template<> struct ConwayPolynomial<883, 8> { using ZPZ = aerobus::zpz<883>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<740>, ZPZV<762>, ZPZV<768>, ZPZV<28; }; //
           NOLINT
05980
                 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<557>, ZPZV<881»;
           }; // NOLINT
05981
                  template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
           POLYV<ZPZV<1>, ZPZV<882»; }; // NOLINT
05982
                  template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
           POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; }; // NOLINT
                  template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
05983
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<882»; }; // NOLINT
05984
                  template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZY<883>, ZPZY<5»; }; // NOLINT template<> struct ConwayPolynomial<887, 5> { using ZPZ = aerobus::zpz<887>; using type =
05985
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<582»; }; // 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»; }; // NOLINT
05986
                  template<> struct ConwayPolynomial<887, 7> { using ZPZ = aerobus::zpz<887>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<82»; };
05988
                 template<> struct ConwayPolynomial<887, 8> { using ZPZ = aerobus::zpz<887>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<781>, ZPZV<381>, ZPZV<706>, ZPZV<5»; }; //
          NOLINT
                  template<> struct ConwayPolynomial<887, 9> { using ZPZ = aerobus::zpz<887>; using type =
05989
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<727>, ZPZV<345>, ZPZV<882»;
           }; // NOLINT
05990
                  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 =
05991
          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 =
05993
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
05994
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; }; // NOLINT
05995
                  template<> struct ConwayPolynomial<907, 6> { using ZPZ = aerobus::zpz<907>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<626>, ZPZV<752>, ZPZV<266>, ZPZV<2»; }; // NOLINT
05996
                 template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<905»; }; // NOLINT template<> struct ConwayPolynomial<907, 8> { using ZPZ = aerobus::zpz<907>; using type =
05997
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<518>, ZPZV<811>, ZPZV<81; //
           NOLINT
                  template<> struct ConwayPolynomial<907, 9> { using ZPZ = aerobus::zpz<907>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<783>, ZPZV<787, ZPZV<905»;
           }; // NOLINT
05999
                  template<> struct ConwayPolynomial<911, 1> { using ZPZ = aerobus::zpz<911>; using type =
          POLYV<ZPZV<1>, ZPZV<894»; }; // NOLINT
06000
                  template<> struct ConwayPolynomial<911, 2> { using ZPZ = aerobus::zpz<911>; using type =
          POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; }; // NOLINT
                  template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
06001
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; }; // NOLINT template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
06002
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
06003
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<894»; };
                                                                                                                                // NOLINT
                  template<> struct ConwayPolynomial<911, 6> { using ZPZ = aerobus::zpz<911>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<683>, ZPZV<19>, ZPZV<17»; }; // NOLINT
06005
                 template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
06006
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<590>, ZPZV<168>, ZPZV<17»; }; //
06007
            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
             template<> struct ConwayPolynomial<919, 1> { using ZPZ = aerobus::zpz<919>; using type =
06008
       POLYV<ZPZV<1>, ZPZV<912»; }; // NOLINT
             template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
06009
       POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; }; // NOLINT
06010
             template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<912»; }; // NOLINT template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
06011
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7»; };
                                                                                    // NOLINT
             template<> struct ConwayPolynomial<919, 5> { using ZPZ = aerobus::zpz<919>; using type =
06012
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<912»; }; // NOLINT
06013
             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
06014
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<9>, ZPZV<9>, ZPZV<9>, ZPZV<9>, ZPZV<9
             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<504>, ZPZV<7»; }; //
        NOLINT
06016
            template<> struct ConwayPolynomial<919, 9> { using ZPZ = aerobus::zpz<919>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<410>, ZPZV<623>, ZPZV<912»;
        }; // NOLINT
             template<> struct ConwayPolynomial<929, 1> { using ZPZ = aerobus::zpz<929>; using type =
       POLYV<ZPZV<1>, ZPZV<926»; }; // NOLINT
06018
             template<> struct ConwayPolynomial<929, 2> { using ZPZ = aerobus::zpz<929>; using type =
       POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3»; }; // NOLINT
             template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
06019
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<926»; }; // NOLINT template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type =
06020
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<787>, ZPZV<3»; }; // NOLINT
06021
            template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<32, ZPZV<926»; }; // NOLINT template<> struct ConwayPolynomial<929, 6> { using ZPZ = aerobus::zpz<929>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<805>, ZPZV<92>, ZPZV<86>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type =
06022
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<926»; };
            template<> struct ConwayPolynomial<929, 8> { using ZPZ = aerobus::zpz<929>; using type =
06024
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699>, ZPZV<292>, ZPZV<586>, ZPZV<586>, ZPZV<5%; }; //
        NOLINT
             template<> struct ConwayPolynomial<929, 9> { using ZPZ = aerobus::zpz<929>; using type =
06025
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<481>, ZPZV<199>, ZPZV<926»;
06026
             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 =
06027
       POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5»; }; // NOLINT
             template<> struct ConwayPolynomial<937, 3> { using ZPZ = aerobus::zpz<937>; using type =
06028
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<932»; }; // NOLINT template<> struct ConwayPolynomial<937, 4> { using ZPZ = aerobus::zpz<937>; using type =
        \verb"POLYV<ZPZV<1>, \ \verb"ZPZV<0>, \ \verb"ZPZV<23>, \ \verb"ZPZV<585>, \ \verb"ZPZV<5"; \ \verb"}; \ \ // \ \verb"NOLINT" 
06030
             template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<932»; }; // NOLINT
             template<> struct ConwayPolynomial<937, 6> { using ZPZ = aerobus::zpz<937>; using type =
06031
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<794>, ZPZV<727>, ZPZV<934>, ZPZV<5»; }; // NOLINT
            template<> struct ConwayPolynomial<937, 7> { using ZPZ = aerobus::zpz<937>; using type
06032
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<2>, ZPZY<23, ZPZY<32»; }; // NOLINT template<> struct ConwayPolynomial<937, 8> { using ZPZ = aerobus::zpz<937>; using type =
06033
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<26>, ZPZV<53>, ZPZV<5»; };
        NOLINT
06034
             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<28>, ZPZV<533>, ZPZV<483>, ZPZV<932»;
        }; // NOLINT
06035
             template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
       POLYV<ZPZV<1>, ZPZV<939»; }; // NOLINT
             template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
06036
       POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2»; }; // NOLINT
             template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
06037
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type =
06038
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2%; }; // NOLINT template<> struct ConwayPolynomial<941, 5> { using ZPZ = aerobus::zpz<941>; using type =
06039
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<39»; }; // NOLINT
             template<> struct ConwayPolynomial<941, 6> { using ZPZ = aerobus::zpz<941>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<459>, ZPZV<694>, ZPZV<538>, ZPZV<2»; }; // NOLINT
       template<> struct ConwayPolynomial<941, 7> { using ZPZ = aerobus::zpz<941>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<939»; }; // NOLINT
template<> struct ConwayPolynomial<941, 8> { using ZPZ = aerobus::zpz<941>; using type =
06041
06042
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<675>, ZPZV<590>, ZPZV<2»; }; //
       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<708>, ZPZV<197>, ZPZV<197
        }; // NOLINT
             template<> struct ConwayPolynomial<947, 1> { using ZPZ = aerobus::zpz<947>; using type =
06044
        POLYV<ZPZV<1>, ZPZV<945»; }; // NOLINT
```

9.3 aerobus.h 205

```
06045
                      template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
            POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2»; }; // NOLINT
06046
                     template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 4> { using ZPZ = aerobus::zpz<947>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2»; }; // NOLINT
06047
                      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
                     template<> struct ConwayPolynomial<947, 6> { using ZPZ = aerobus::zpz<947>; using type =
06049
            POLYY<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 =
06050
            POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 8> { using ZPZ = aerobus::zpz<947>; using type =
06051
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<845>, ZPZV<597>, ZPZV<581>, ZPZV<2»; };
             NOLINT
06052
                     template<> struct ConwayPolynomial<947, 9> { using ZPZ = aerobus::zpz<947>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<26>, ZPZV<808>, ZPZV<945»;
             }; // NOLINT
06053
                      template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
             POLYV<ZPZV<1>, ZPZV<950»; }; // NOLINT
                      template<> struct ConwayPolynomial<953, 2> { using ZPZ = aerobus::zpz<953>; using type =
             POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3»; }; // NOLINT
                     template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
06055
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950»; }; // NOLINT template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
06056
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<865>, ZPZV<3»; }; // NOLINT
                     template<> struct ConwayPolynomial<953, 5> { using ZPZ = aerobus::zpz<953>; using type =
06057
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<950»; }; // NOLINT
06058
                      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 template<> struct ConwayPolynomial<953, 7> { using ZPZ = aerobus::zpz<953>; using type
06059
            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
             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 =
06061
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<819>, ZPZV<316>, ZPZV<950»;
06062
                      template<> struct ConwayPolynomial<967, 1> { using ZPZ = aerobus::zpz<967>; using type =
             POLYV<ZPZV<1>, ZPZV<962»; }; // NOLINT
06063
                     template<> struct ConwayPolynomial<967, 2> { using ZPZ = aerobus::zpz<967>; using type =
            POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
06064
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962»; }; // NOLINT
                      template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; }; // NOLINT
06066
                     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
                     template<> struct ConwayPolynomial<967, 6> { using ZPZ = aerobus::zpz<967>; using type =
06067
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<948>, ZPZV<831>, ZPZV<5»; }; // NOLINT
06068
                     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<9>, ZPZV<9>, ZPZV<9>, ZPZV<9+, ZPZV<962*; }; //
06069
                     template<> struct ConwayPolynomial<967, 8> { using ZPZ = aerobus::zpz<967>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<840>, ZPZV<502>, ZPZV<136>, ZPZV<5»; }; //
             NOLINT
06070
            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<51>, ZPZV<512>, ZPZV<783>, ZPZV<962»;</pre>
            }; // NOLINT
                      template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
06071
            POLYV<ZPZV<1>, ZPZV<965»; }; // NOLINT
                     template<> struct ConwayPolynomial<971, 2> { using ZPZ = aerobus::zpz<971>; using type =
06072
            POLYV<ZPZV<1>, ZPZV<970>, ZPZV<6»; }; // NOLINT
06073
                      template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
            POLYY<ZPZY<1>, ZPZY<0>, ZPZY<3>, ZPZV<965»; }; // NOLINT template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
06074
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<527>, ZPZV<6»; }; // NOLINT
                     template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
06075
            POLYY<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 =
06076
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<970>, ZPZV<729>, ZPZV<718>, ZPZV<6»; }; // NOLINT
                     template<> struct ConwayPolynomial<971, 7> { using ZPZ = aerobus::zpz<971>; using type
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<965»; };
06078
                     template<> struct ConwayPolynomial<971, 8> { using ZPZ = aerobus::zpz<971>; using type =
             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<2D>, ZPZV<2D , ZPZV<
             NOLINT
                     template<> struct ConwayPolynomial<971, 9> { using ZPZ = aerobus::zpz<971>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
             }; // NOLINT
06080
                     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 =
06081
            POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; }; // NOLINT
                      template<> struct ConwayPolynomial<977, 3> { using ZPZ = aerobus::zpz<977>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<974»; };
                                                                                                                   // NOLINT
                    template<> struct ConwayPolynomial<977, 4> { using ZPZ = aerobus::zpz<977>; using type =
06083
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<800>, ZPZV<30, ZPZV<30, ZPZV<30; ZPZV
06084
```

206 File Documentation

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<974»; };
         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 =
06086
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<974*; }; // NOLINT template<> struct ConwayPolynomial<977, 8> { using ZPZ = aerobus::zpz<977>; using type =
06087
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<85>, ZPZV<807>, ZPZV<77>, ZPZV<3»; };
         NOLINT
06088
              template<> struct ConwayPolynomial<977, 9> { using ZPZ = aerobus::zpz<977>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<740>, ZPZV<740
         }; // NOLINT
06089
                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 =
         POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; }; // NOLINT
06091
              template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978»; }; // NOLINT template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
06092
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
06093
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<978»; }; // NOLINT
06094
               template<> struct ConwayPolynomial<983, 6> { using ZPZ = aerobus::zpz<983>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<849>, ZPZV<296>, ZPZV<228>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type
06095
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<978»; }; // NOLINT
               template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<53»; }; //
         NOLINT
         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<858>, ZPZV<87>, ZPZV<978»;</pre>
06097
         }; // NOLINT
06098
                template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
         POLYV<ZPZV<1>, ZPZV<985»; }; // NOLINT
06099
              template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
         POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; }; // NOLINT
               template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
06100
         POLYV<ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<985»; }; // NOLINT template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6»; }; // NOLINT
              template<> struct ConwayPolynomial<991, 5> { using ZPZ = aerobus::zpz<991>; using type =
06102
         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 =
06103
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<637>, ZPZV<855>, ZPZV<278>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<991, 7> { using ZPZ = aerobus::zpz<991>; using type
06104
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<985»; }; // NOLINT
06105
               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
06106
               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<46>, ZPZV<46>, ZPZV<222>, ZPZV<985»;
         }; // NOLINT
   template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
06107
         POLYV<ZPZV<1>, ZPZV<990»; }; // NOLINT template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
06108
         POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; }; // NOLINT
               template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
06109
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990»; }; // NOLINT
              template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
06110
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<62>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
06111
         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<9>, ZPZV<981>, ZPZV<58>, ZPZV<260>, ZPZV<7»; }; // NOLINT
06112
               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<1>, ZPZV<990»; };
06114
              template<> struct ConwayPolynomial<997, 8> { using ZPZ = aerobus::zpz<997>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<934>, ZPZV<473>, ZPZV<241>, ZPZV<241>, ZPZV<7»; }; //
         NOLINT
              template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<732>, ZPZV<616>, ZPZV<990»;
          }; // NOLINT
06116 #endif // DO NOT DOCUMENT
06117 } // namespace aerobus
06118 #endif // AEROBUS_CONWAY_IMPORTS
06120 #endif // __INC_AEROBUS__ // NOLINT
```

src/examples.h File Reference 9.4

examples.h 9.5

Go to the documentation of this file.

00001 #ifndef SRC_EXAMPLES_H_

00002 #define SRC_EXAMPLES_H_

00050 #endif // SRC_EXAMPLES_H_

208 File Documentation

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)));
```

211

```
};
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 inject_ring_t, 60 inner_type, 60	aerobus::known_polynomials, 40 hermite_kind, 40 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
	gt t, 75
pos_v, 62	<u> </u>
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 acrohyeris prime $\langle n \rangle$ 71	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::zpz<p>::val<math><x>, 100
aerobus::Quotient < Ring, X >, 81
     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
```

corobus inclumental < Ping > 76	make a64 t
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 $<$ p $>$, 109
aerobus::i64, 69	mulfractions t
	-
aerobus::polynomial < Ring >, 80	aerobus, 29
aerobus::zpz, 111	ono
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 $, 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::val< x >, 102	pos_t
40.0000	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
It_t	aerobus::i32, 62
aerobus::i32, 60	aerobus::i64, 70
	aerobus::polynomial< Ring >, 80
aerobus::i64, 67	aerobus::Quotient< Ring, X >, 85
aerobus::polynomial < Ring >, 76	aerobus::zpz $<$ p $>$, 112
aerobus::zpz, 109	
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, 31	aerobus::ContinuedFraction< a0 >, 47 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 index > 0) > >, 45
aerobus, 31	
SQRT2 fraction	aerobus::polynomial< Ring >::val< coeffN
aerobus, 31	>::coeff_at< index, std::enable_if_t<(index==0):
SQRT3_fraction	>, 46
aerobus, 31	aerobus::Quotient < Ring, X >::val < V >, 100
src/aerobus.h, 113	aerobus::type_list< Ts >::pop_front, 81
src/examples.h, 207	type_at_t
stirling_1_signed_t	aerobus::internal, 40
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
aerobus:: $zpz $, 110	• •
4010000.12P2 , 110	>, 97
tail	vmul_t
aerobus::type_list< Ts >::pop_front, 81	aerobus, 34
aerobus::type list< Ts >::split< index >, 86	X
tan	aerobus::polynomial< Ring >, 79
aerobus, 33	acrobadpolyflormal < rung > , 70
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, 110
aerobus::i64::val< x >, 94	1 10 10 10 10 10 10 10 10 10 10 10 10 10
aerobus::polynomial < Ring >::val < coeffN >, 104	
aerobus::polynomial < Ring >::val < coeffN, coeffs	
>, 98	
- ,	