Aerobus

v1.2

Generated by Doxygen 1.9.8

Chapter 1

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) -I<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.

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. Then move to top directory :

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

results on my laptop :

```
./benchmarks_avx512.exe
[std math] 5.358e-01 Gsin/s
[std fast math] 3.389e+00 Gsin/s
[aerobus deg 1] 1.871e+01 Gsin/s
average error (vs std) : 4.36e-02
max error (vs std) : 1.50e-01
[aerobus deg 3] 1.943e+01 Gsin/s
average error (vs std) : 1.85e-04
max error (vs std) : 8.17e-04
[aerobus deg 5] 1.335e+01 Gsin/s
average error (vs std) : 6.07e-07
max error (vs std) : 3.63e-06
[aerobus deg 7] 8.634e+00 Gsin/s
average error (vs std) : 1.27e-09
max error (vs std) : 9.75e-09
[aerobus deg 9] 6.171e+00 Gsin/s
average error (vs std) : 1.89e-12
max error (vs std) : 1.78e-11
[aerobus deg 11] 4.731e+00 Gsin/s
average error (vs std) : 2.12e-15
max error (vs std) : 2.40e-14
[aerobus deg 13] 3.862e+00 Gsin/s
average error (vs std) : 3.16e-17
max error (vs std) : 3.33e-16
[aerobus deg 15] 3.359e+00 Gsin/s
average error (vs std) : 3.13e-17
max error (vs std) : 3.33e-16
[aerobus deg 17] 2.947e+00 Gsin/s
average error (vs std) : 3.13e-17
max error (vs std) : 3.33e-16
average error (vs std) : 3.13e-17
max error (vs std) : 3.33e-16
```

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<p> : integers modulo p (prime number) on 32 bits

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

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

and the following "elements" :

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

1.2.2 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) {...}
```

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
- laguerre
- 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>;
```

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↩ 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.

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

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

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

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

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

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

Yields this assembly (clang 17, -mavx2 -O3) where we can see a pile of Fused-Multiply-Add vector instructions, generated because we unrolled completely the Horner evaluation loop:

```
compute_expm1(unsigned long, double const*, double*):
  lea     rax, [rdi-1]
  cmp     rax, 2
  jbe     .L5
  mov     rcx, rdi
  xor     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]
  sal     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    ymm5, QWORD PTR .LC19[rip]
  vbroadcastsd    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
  vfmadd132pd     ymm0, ymm8, ymm15
  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
  add     rax, 32
  cmp     rcx, rax
  jne     .L3
  mov     rax, rdi
  and     rax, -4
  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`.

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

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

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

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

For example, we can easily define `rational functions` by taking the ring of fractions of polynomials:

```
using namespace aerobus;
using RF64 = FractionField<polynomial<q64>>;
```

Which also have an evaluation function, as polynomial do.

1.3.2 Quotient

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

For example, if we want `R` to be $\mathbb{Z}$ represented as `aerobus::i64`, we can express arithmetic modulo 17 using:

```
using namespace aerobus;
using ZpZ = Quotient<i64, i64::val<17>>;
```

As we could have using `zpz<17>`.

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

1.4 Misc

1.4.1 Continued Fractions

Aerobus gives an implementation for continued fractions. It can be used this way:

```
using namespace aerobus;
using T = ContinuedFraction<1,2,3,4>;
constexpr double x = T::val;
```

As practical examples, aerobus gives continued fractions of π, e, $\sqrt{2}$ and $\sqrt{3}$:

```
constexpr double A_SQRT3 = aerobus::SQRT3_fraction::val; // 1.7320508075688772935
```

1.5 CUDA

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

Unfortunately, NVIDIA did not put enough constexpr in its 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 this bug.

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

Please push to make these bug fixed by NVIDIA.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Concept Index

3.1 Concepts

Here is a list of all concepts with brief descriptions:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

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 multiple values (v1 + v2 + ... + vn) vals must have same "enclosing_type" and "enclosing_type" must have an mul_t binary operator
- template<typename val >
  using abs_t = std::conditional_t< val::enclosing_type::template pos_v< val >, val, typename val::enclosing↩
  _type::template sub_t< typename val::enclosing_type::zero, val > >

  computes absolute value of 'val' val must be a 'value' in a Ring satisfying 'IsEuclideanDomain' concept
- template<typename Ring >
  using FractionField = typename internal::FractionFieldImpl< Ring >::type
- 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 denominator)
- using q64 = FractionField< i64 >

  64 bits rationals rationals with 64 bits numerator and denominator
- using pi64 = polynomial< i64 >

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

  polynomial with 64 bits rationals coefficients
- using fpq64 = FractionField< polynomial< q64 > >

  polynomial with 64 bits rational coefficients
- template<typename Ring , typename v1 , typename v2 >
  using makefraction_t = typename FractionField< Ring >::template val< v1, v2 >

  helper type : the rational V1/V2 in the field of fractions of Ring
- template<typename v >
  using embed_int_poly_in_fractions_t = typename Embed< polynomial< typename v::ring_type >, polynomial< FractionField< typename v::ring_type > > >::template type< v >

  embed a polynomial with integers coefficients into rational coefficients polynomials
- template<int64_t p, int64_t q>
  using make_q64_t = typename q64::template simplify_t< typename q64::val< i64::inject_constant_t< p >, i64::inject_constant_t< q > > >

  helper type : make a fraction from numerator and denominator
- template<int32_t p, int32_t q>
  using make_q32_t = typename q32::template simplify_t< typename q32::val< i32::inject_constant_t< p >, i32::inject_constant_t< q > > >

  helper type : make a fraction from numerator and denominator
- template<typename Ring , typename v1 , typename v2 >
  using addfractions_t = typename FractionField< Ring >::template add_t< v1, v2 >

  helper type : adds two fractions
- template<typename Ring , typename v1 , typename v2 >
  using mulfractions_t = typename FractionField< Ring >::template mul_t< v1, v2 >

  helper type : multiplies two fractions
- template<typename Ring , auto... xs>
  using make_int_polynomial_t = typename polynomial< Ring >::template val< typename Ring::template inject_constant_t< xs >... >

  make a polynomial with coefficients in Ring
- template<typename Ring , auto... xs>
  using make_frac_polynomial_t = typename polynomial< FractionField< Ring > >::template val< typename FractionField< Ring >::template inject_constant_t< xs >... >

  make a polynomial with coefficients in FractionField<Ring>
- template<typename T , size_t i>
  using factorial_t = typename internal::factorial< T, i >::type

  computes factorial(i), as type
- template<typename T , size_t k, size_t n>
  using combination_t = typename internal::combination< T, k, n >::type

computes binomial coefficient (k among n) as type

- template<typename T , size_t n>
  using bernoulli_t = typename internal::bernoulli< T, n >::type

  nth bernoulli number as type in T
- template<typename T , size_t n>
  using bell_t = typename internal::bell_helper< T, n >::type

  Bell numbers.
- template<typename T , int k>
  using alternate_t = typename internal::alternate< T, k >::type

  (-1)^k as type in T
- template<typename T , int n, int k>
  using stirling_signed_t = typename internal::stirling_helper< T, n, k >::type

  Stirling number of first king (signed) – as types.
- template<typename T , int n, int k>
  using stirling_unsigned_t = abs_t< typename internal::stirling_helper< T, n, k >::type >

  Stirling number of first king (unsigned) – as types.
- template<typename T , typename p , size_t n>
  using pow_t = typename internal::pow< T, p, n >::type

  p^n (as 'val' type in T)
- template<typename T , template< typename, size_t index > typename coeff_at, size_t deg>
  using taylor = typename internal::make_taylor_impl< T, coeff_at, internal::make_index_sequence_reverse< deg+1 > >::type
- template<typename Integers , size_t deg>
  using exp = taylor< Integers, internal::exp_coeff, deg >

  e^x
- template<typename Integers , size_t deg>
  using expm1 = typename polynomial< FractionField< Integers > >::template sub_t< exp< Integers, deg >, typename polynomial< FractionField< Integers > >::one >

  $e^x - 1$
- template<typename Integers , size_t deg>
  using lnp1 = taylor< Integers, internal::lnp1_coeff, deg >

  $\ln(1+x)$
- template<typename Integers , size_t deg>
  using atan = taylor< Integers, internal::atan_coeff, deg >

  $\arctan(x)$
- template<typename Integers , size_t deg>
  using sin = taylor< Integers, internal::sin_coeff, deg >

  $\sin(x)$
- template<typename Integers , size_t deg>
  using sinh = taylor< Integers, internal::sh_coeff, deg >

  $\sinh(x)$
- template<typename Integers , size_t deg>
  using cosh = taylor< Integers, internal::cosh_coeff, deg >

  $\cosh(x)$ *hyperbolic cosine*
- template<typename Integers , size_t deg>
  using cos = taylor< Integers, internal::cos_coeff, deg >

  $\cos(x)$ *cosinus*
- template<typename Integers , size_t deg>
  using geometric_sum = taylor< Integers, internal::geom_coeff, deg >

  $\frac{1}{1-x}$ *zero development of* $\frac{1}{1-x}$
- template<typename Integers , size_t deg>
  using asin = taylor< Integers, internal::asin_coeff, deg >

  $\arcsin(x)$ *arc sinus*

- template<typename Integers , size_t deg>
  using asinh = taylor< Integers, internal::asinh_coeff, deg >
  $\operatorname{arcsinh}(x)$ *arc hyperbolic sinus*
- template<typename Integers , size_t deg>
  using atanh = taylor< Integers, internal::atanh_coeff, deg >
  $\operatorname{arctanh}(x)$ *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 >
- using SQRT2_fraction = ContinuedFraction< 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 >
  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 >
  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

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

6.1.1 Detailed Description

main namespace for all publicly exposed types or functions

6.1.2 Typedef Documentation

6.1.2.1 abs_t

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

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

Template Parameters

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

6.1.2.2 add_t

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

generic addition

Template Parameters

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

6.1.2.3 addfractions_t

```
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

Ring	
v1	belongs to FractionField<Ring>
v2	belongs to FranctionField<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>
```

$\text{arcsinh}(x)$ 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>
```

$\text{arctanh}(x)$ 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>
```

$\cosh(x)$ 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

X	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>
```

6.1.2.16 embed_int_poly_in_fractions_t

```
template<typename v >
using aerobus::embed_int_poly_in_fractions_t = typedef 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

Lives in polynomial<FractionField<Ring>>

Template Parameters

Ring	Integers
a	value in polynomial<Ring>

6.1.2.17 exp

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

e^x

Template Parameters

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

6.1.2.18 expm1

```
template<typename Integers , size_t deg>
using aerobus::expm1 = typedef typename polynomial<FractionField<Integers> >::template sub_t<
exp<Integers, deg>, typename polynomial<FractionField<Integers> >::one>
```

$e^x - 1$

Template Parameters

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

6.1.2.19 factorial_t

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

computes factorial(i), as type

Template Parameters

T	Ring type (e.g. i32)
i	

6.1.2.20 fpq32

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

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

6.1.2.21 fpq64

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

polynomial with 64 bits rational coefficients

6.1.2.22 FractionField

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

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>
```

$\frac{1}{1-x}$ zero development of $\frac{1}{1-x}$

Template Parameters

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

6.1.2.25 lnp1

```
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> >↩
::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

p	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

p	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

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

6.1.2.32 mulfractions_t

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

helper type : multiplies two fractions

Template Parameters

Ring	
v1	belongs to FractionField<Ring>
v2	belongs to FranctionField<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>
```

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)
p	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

```
using aerobus::SQRT2_fraction = typedef ContinuedFraction<1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2>
```

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>
```

approximation of

6.1.2.43 stirling_signed_t

```
template<typename T , int n, int k>
using aerobus::stirling_signed_t = typedef typename internal::stirling_helper<T, n, k>::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_unsigned_t

```
template<typename T , int n, int k>
using aerobus::stirling_unsigned_t = typedef abs_t<typename internal::stirling_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 sub_t

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

generic subtraction

Template Parameters

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

6.1.2.46 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

6.1.2.47 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

6.1.2.48 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_at	- implementation giving the 'value' (seen as type in FractionField<T>
deg	

6.1.2.49 vadd_t

```
template<typename...  vals>
using aerobus::vadd_t = typedef 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 Parameters

...vals	

6.1.2.50 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

...vals	

6.1.3 Function Documentation

6.1.3.1 aligned_malloc()

```
template<typename T >
T * aerobus::aligned_malloc (
             size_t count,
             size_t alignment )
```

'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<FloatType>
[inline], [constexpr]
```

nth bernoulli number as value in FloatType

Template Parameters

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

6.1.4.3 combination_v

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

computes binomial coefficients (k among n) as value

Template Parameters

T	(aerobus::i32 for example)
k	
n	

6.1.4.4 factorial_v

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

computes factorial(i) as value in T

Template Parameters

T	(aerobus::i64 for example)
i	

6.2 aerobus::internal Namespace Reference

internal implementations, subject to breaking changes without notice

Classes

- struct **_FractionField**
- struct **_FractionField**< **Ring, std::enable_if_t**< **Ring::is_euclidean_domain** > >
- struct **_is_prime**
- struct **_is_prime**< **0, i** >
- struct **_is_prime**< **1, i** >
- struct **_is_prime**< **2, i** >
- struct **_is_prime**< **3, i** >
- struct **_is_prime**< **5, i** >
- struct **_is_prime**< **7, i** >
- struct **_is_prime**< **n, i, std::enable_if_t**<(**n !=2 &&n !=3 &&n % 2 !=0 &&n % 3==0**)> >
- struct **_is_prime**< **n, i, std::enable_if_t**<(**n !=2 &&n % 2==0**)> >
- struct **_is_prime**< **n, i, std::enable_if_t**<(**n % i==0 &&n** >**=9 &&n % 3 !=0 &&n % 2 !=0 &&i** ∗**i** > **n**)> >
- struct **_is_prime**< **n, i, std::enable_if_t**<(**n %(i+2) !=0 &&n % i !=0 &&n** >**=9 &&n % 3 !=0 &&n % 2 !=0 &&(i** ∗**i**<**=n)**)> >
- struct **_is_prime**< **n, i, std::enable_if_t**<(**n %(i+2)==0 &&n** >**=9 &&n % 3 !=0 &&n % 2 !=0 &&i** ∗**i**<**=n**)> >
- struct **_is_prime**< **n, i, std::enable_if_t**<(**n** >**=9 &&i** ∗**i** > **n**)> >
- struct **alternate**
- struct **alternate**< **T, k, std::enable_if_t**< **k % 2 !=0** > >
- struct **alternate**< **T, k, std::enable_if_t**< **k % 2==0** > >
- struct **asin_coeff**
- struct **asin_coeff_helper**
- struct **asin_coeff_helper**< **T, i, std::enable_if_t**<(**i &1)==0** > >
- struct **asin_coeff_helper**< **T, i, std::enable_if_t**<(**i &1)==1** > >
- struct **asinh_coeff**
- struct **asinh_coeff_helper**
- struct **asinh_coeff_helper**< **T, i, std::enable_if_t**<(**i &1)==0** > >
- struct **asinh_coeff_helper**< **T, i, std::enable_if_t**<(**i &1)==1** > >
- struct **atan_coeff**
- struct **atan_coeff_helper**
- struct **atan_coeff_helper**< **T, i, std::enable_if_t**<(**i &1)==0** > >

- struct **atan_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >**
- struct **atanh_coeff**
- struct **atanh_coeff_helper**
- struct **atanh_coeff_helper< T, i, std::enable_if_t<(i &1)==0 > >**
- struct **atanh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >**
- struct **bell_helper**
- struct **bell_helper< T, 0 >**
- struct **bell_helper< T, 1 >**
- struct **bell_helper< T, n, std::enable_if_t<(n > 1)> >**
- struct **bernoulli**
- struct **bernoulli< T, 0 >**
- struct **bernoulli_coeff**
- struct **bernoulli_helper**
- struct **bernoulli_helper< T, accum, m, m >**
- struct **bernstein_helper**
- struct **bernstein_helper< 0, 0, l >**
- struct **bernstein_helper< i, m, l, std::enable_if_t<(m > 0) &&(i > 0) &&(i< m)> >**
- struct **bernstein_helper< i, m, l, std::enable_if_t<(m > 0) &&(i==0)> >**
- struct **bernstein_helper< i, m, l, std::enable_if_t<(m > 0) &&(i==m)> >**
- struct **chebyshev_helper**
- struct **chebyshev_helper< 1, 0, l >**
- struct **chebyshev_helper< 1, 1, l >**
- struct **chebyshev_helper< 2, 0, l >**
- struct **chebyshev_helper< 2, 1, l >**
- struct **combination**
- struct **combination_helper**
- struct **combination_helper< T, 0, n >**
- struct **combination_helper< T, k, n, std::enable_if_t<(n >=0 &&k >(n/2) &&k > 0)> >**
- struct **combination_helper< T, k, n, std::enable_if_t<(n >=0 &&k<=(n/2) &&k > 0)> >**
- struct **cos_coeff**
- struct **cos_coeff_helper**
- struct **cos_coeff_helper< T, i, std::enable_if_t<(i &1)==0 > >**
- struct **cos_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >**
- struct **cosh_coeff**
- struct **cosh_coeff_helper**
- struct **cosh_coeff_helper< T, i, std::enable_if_t<(i &1)==0 > >**
- struct **cosh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >**
- struct **exp_coeff**
- struct **factorial**
- struct **factorial< T, 0 >**
- struct **factorial< T, x, std::enable_if_t<(x > 0)> >**
- struct **fma_helper**
- struct **fma_helper< double >**
- struct **fma_helper< float >**
- struct **fma_helper< int16_t >**
- struct **fma_helper< int32_t >**
- struct **fma_helper< int64_t >**
- struct **FractionFieldImpl**
- struct **FractionFieldImpl< Field, std::enable_if_t< Field::is_field > >**
- struct **FractionFieldImpl< Ring, std::enable_if_t<!Ring::is_field > >**
- struct **gcd**

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

- struct **geom_coeff**
- struct **hermite_helper**
- struct **hermite_helper< 0, known_polynomials::hermite_kind::physicist, I >**
- struct **hermite_helper< 0, known_polynomials::hermite_kind::probabilist, I >**
- struct **hermite_helper< 1, known_polynomials::hermite_kind::physicist, I >**
- struct **hermite_helper< 1, known_polynomials::hermite_kind::probabilist, I >**
- struct **hermite_helper< deg, known_polynomials::hermite_kind::physicist, I >**
- struct **hermite_helper< deg, known_polynomials::hermite_kind::probabilist, I >**
- 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 **lnp1_coeff**
- struct **lnp1_coeff< T, 0 >**
- struct **make_taylor_impl**
- struct **make_taylor_impl< T, coeff_at, std::integer_sequence< size_t, Is... > >**
- struct **pop_front_h**
- struct **pow**
- struct **pow< T, p, n, std::enable_if_t< n==0 > >**
- struct **pow< T, p, n, std::enable_if_t<(n % 2==1)> >**
- struct **pow< T, p, n, std::enable_if_t<(n > 0 &&n % 2==0)> >**
- struct **pow_scalar**
- struct **remove_h**
- struct **sh_coeff**
- struct **sh_coeff_helper**
- struct **sh_coeff_helper< T, i, std::enable_if_t<(i &1)==0 > >**
- struct **sh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >**
- struct **sin_coeff**
- struct **sin_coeff_helper**
- struct **sin_coeff_helper< T, i, std::enable_if_t<(i &1)==0 > >**
- struct **sin_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >**
- struct **split_h**
- struct **split_h< 0, L1, L2 >**
- struct **staticcast**
- struct **stirling_helper**
- struct **stirling_helper< T, 0, 0 >**
- struct **stirling_helper< T, 0, n, std::enable_if_t<(n > 0)> >**
- struct **stirling_helper< T, n, 0, std::enable_if_t<(n > 0)> >**
- struct **stirling_helper< T, n, k, std::enable_if_t<(k > 0) &&(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 **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... Is>
  constexpr auto index_sequence_reverse (std::index_sequence< Is... > 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()

```
template<std::size_t... Is>
constexpr auto aerobus::internal::index_sequence_reverse (
            std::index_sequence< Is... > const & ) -> decltype(std::index_sequence< sizeof...(Is)
- 1U - Is... >{}) [constexpr]
```

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

Typedefs

- template<size_t deg, typename I = aerobus::i64>
  using chebyshev_T = typename internal::chebyshev_helper< 1, deg, I >::type

  Chebyshev polynomials of first kind.
- template<size_t deg, typename I = aerobus::i64>
  using chebyshev_U = typename internal::chebyshev_helper< 2, deg, I >::type

  Chebyshev polynomials of second kind.
- template<size_t deg, typename I = aerobus::i64>
  using laguerre = typename internal::laguerre_helper< deg, I >::type

  Laguerre polynomials.
- template<size_t deg, typename I = aerobus::i64>
  using hermite_prob = typename internal::hermite_helper< deg, hermite_kind::probabilist, I >::type

  Hermite polynomials - probabilist form.
- template<size_t deg, typename I = aerobus::i64>
  using hermite_phys = typename internal::hermite_helper< deg, hermite_kind::physicist, I >::type

  Hermite polynomials - physicist form.
- template<size_t i, size_t m, typename I = aerobus::i64>
  using bernstein = typename internal::bernstein_helper< i, m, I >::type

  Bernstein polynomials.
- template<size_t deg, typename I = aerobus::i64>
  using legendre = typename internal::legendre_helper< deg, I >::type

  Legendre polynomials.
- template<size_t deg, typename I = aerobus::i64>
  using bernoulli = taylor< I, internal::bernoulli_coeff< deg >::template inner, deg >

  Bernoulli polynomials.

Enumerations

- enum hermite_kind { probabilist , physicist }

6.3.1 Detailed Description

families of well known polynomials such as Hermite or Bernstein

6.3.2 Typedef Documentation

6.3.2.1 bernoulli

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::bernoulli = typedef taylor<I, internal::bernoulli_coeff<deg>↩
::template inner, deg>
```

Bernoulli polynomials.

Lives in polynomial<FractionField<I>>

See also

See in Wikipedia

Template Parameters

deg	degree of polynomial
I	Integers ring (defaults to aerobus::i64)

6.3.2.2 bernstein

```
template<size_t i, size_t m, typename I = aerobus::i64>
using aerobus::known_polynomials::bernstein = typedef typename internal::bernstein_helper<i,
m, I>::type
```

Bernstein polynomials.

Lives in polynomial

See also

See in Wikipedia

Template Parameters

i	*index of polynomial (between 0 and m)*
m	*degree of polynomial*
I	*Integers ring (defaults to aerobus::i64)*

6.3.2.3 chebyshev_T

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::chebyshev_T = typedef typename internal::chebyshev_helper<1,
deg, I>::type
```

Chebyshev polynomials of first kind.

See also

See in Wikipedia

Template Parameters

deg	degree of polynomial
integer	rings (defaults to aerobus::i64)

6.3.2.4 chebyshev_U

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::chebyshev_U = typedef typename internal::chebyshev_helper<2,
deg, I>::type
```

Chebyshev polynomials of second kind.

Lives in polynomial

See also

See in Wikipedia

Template Parameters

deg	*degree of polynomial*
integer	*rings (defaults to aerobus::i64)*

6.3.2.5 hermite_phys

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::hermite_phys = typedef typename internal::hermite_helper<deg,
hermite_kind::physicist, I>::type
```

Hermite polynomials - physicist form.

See also

See in Wikipedia

Template Parameters

deg	degree of polynomial

6.3.2.6 hermite_prob

```
template<size_t deg, typename I = aerobus::i64>
```

```
using aerobus::known_polynomials::hermite_prob = typedef typename internal::hermite_helper<deg,
hermite_kind::probabilist, I>::type
```

Hermite polynomials - probabilist form.

See also

See in Wikipedia

Template Parameters

deg	degree of polynomial

6.3.2.7 laguerre

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::laguerre = typedef typename internal::laguerre_helper<deg,
I>::type
```

Laguerre polynomials.

Lives in polynomial<FractionField<I>>

See also

See in Wikipedia

Template Parameters

deg	degree of polynomial
I	Integers ring (defaults to aerobus::i64)

6.3.2.8 legendre

```
template<size_t deg, typename I = aerobus::i64>
using aerobus::known_polynomials::legendre = typedef typename internal::legendre_helper<deg,
I>::type
```

Legendre polynomials.

Lives in polynomial<FractionField<I>>

See also

See in Wikipedia

Template Parameters

deg	degree of polynomial
I	Integers Ring (defaults to aerobus::i64)

6.3.3 Enumeration Type Documentation

6.3.3.1 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_v<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

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

7.2.2 Detailed Description

Concept to express R is a field.

7.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring.

`#include <aerobus.h>`

7.3.1 Concept definition

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

7.3.2 Detailed Description

Concept to express R is a Ring.

Chapter 8

Class Documentation

8.1 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E > Struct Template Reference

`#include <aerobus.h>`

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

- src/aerobus.h

8.2 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0||index > 0)> > Struct Template Reference

`#include <aerobus.h>`

Public Types

- using type = typename Ring::zero

8.2.1 Member Typedef Documentation

8.2.1.1 type

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

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

- src/aerobus.h

8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference

`#include <aerobus.h>`

Public Types

- using type = aN

8.3.1 Member Typedef Documentation

8.3.1.1 type

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

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

- src/aerobus.h

8.4 aerobus::ContinuedFraction< values > Struct Template Reference

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

`#include <aerobus.h>`

8.4.1 Detailed Description

template<int64_t... values>
struct aerobus::ContinuedFraction< values >

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

Template Parameters

...values	are int64_t

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

- src/aerobus.h

8.5 aerobus::ContinuedFraction< a0 > Struct Template Reference

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

`#include <aerobus.h>`

Public Types

- using type = typename q64::template inject_constant_t< a0 >
  represented value as aerobus::q64

Static Public Attributes

- static constexpr double val = static_cast<double>(a0)
  represented value as double

8.5.1 Detailed Description

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

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

Template Parameters

a0	an integer int64_t

8.5.2 Member Typedef Documentation

8.5.2.1 type

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

represented value as aerobus::q64

8.5.3 Member Data Documentation

8.5.3.1 val

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

represented value as double

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

- src/aerobus.h

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

specialization for multiple coefficients (strictly more than one)

`#include <aerobus.h>`

Public Types

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

  represented value as aerobus::q64

Static Public Attributes

- static constexpr double val = type::template get<double>()

  reprensented value as double

8.6.1 Detailed Description

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

specialization for multiple coefficients (strictly more than one)

Template Parameters

a0	integer (int64_t)
...rest	integers (int64_t)

8.6.2 Member Typedef Documentation

8.6.2.1 type

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

represented value as aerobus::q64

8.6.3 Member Data Documentation

8.6.3.1 val

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

reprensented value as double

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

- src/aerobus.h

8.7 aerobus::ConwayPolynomial Struct Reference

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

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

- src/aerobus.h

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

embedding - struct forward declaration

8.8.1 Detailed Description

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

embedding - struct forward declaration

Template Parameters

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

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

- src/aerobus.h

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

embeds i32 into i64

`#include <aerobus.h>`

Public Types

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

8.9.1 Detailed Description

embeds i32 into i64

8.9.2 Member Typedef Documentation

8.9.2.1 type

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

the i64 representation of val

Template Parameters

val	a value in i32

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

- src/aerobus.h

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

embeds polynomial<Small> into polynomial<Large>

`#include <aerobus.h>`

Public Types

- template<typename v >
  using type = typename at_low< v, typename internal::make_index_sequence_reverse< v::degree+1 > >↩
  ::type
  the polynomial<Large> reprensentation of v

8.10.1 Detailed Description

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

embeds polynomial<Small> into polynomial<Large>

Template Parameters

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

8.10.2 Member Typedef Documentation

8.10.2.1 type

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

the polynomial<Large> reprensentation of v

Template Parameters

v	a value in polynomial<Small>

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

- src/aerobus.h

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

embeds q32 into q64

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

Public Types

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

8.11.1 Detailed Description

embeds q32 into q64

8.11.2 Member Typedef Documentation

8.11.2.1 type

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

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

embeds Quotient<Ring, X> into Ring

`#include <aerobus.h>`

Public Types

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

8.12.1 Detailed Description

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

embeds Quotient<Ring, X> into Ring

Template Parameters

Ring	a Euclidean ring
X	a value in Ring

8.12.2 Member Typedef Documentation

8.12.2.1 type

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

Ring reprensentation of val.

Template Parameters

val	a value in Quotient<Ring, X>

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

- src/aerobus.h

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

embeds values from Ring to its field of fractions

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

Public Types

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

8.13.1 Detailed Description

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

embeds values from Ring to its field of fractions

Template Parameters

Ring	an integers ring, such as i32

8.13.2 Member Typedef Documentation

8.13.2.1 type

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

FractionField<Ring> reprensentation of v.

Template Parameters

v	a Ring value

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

- src/aerobus.h

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

embeds zpz values into i32

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

Public Types

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

8.14.1 Detailed Description

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

embeds zpz values into i32

Template Parameters

x	an integer

8.14.2 Member Typedef Documentation

8.14.2.1 type

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

the i32 reprensentation of val

Template Parameters

val	a value in zpz<x>

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

- src/aerobus.h

8.15 aerobus::i32 Struct Reference

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

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

Classes

- struct val
  values in i32, again represented as types

Public Types

- using inner_type = int32_t
- using zero = val< 0 >
  constant zero
- using one = val< 1 >
  constant one
- template<auto x>
  using inject_constant_t = val< static_cast< int32_t >(x)>
- template<typename v >
  using inject_ring_t = v
- template<typename v1 , typename v2 >
  using add_t = typename add< v1, v2 >::type
- template<typename v1 , typename v2 >
  using sub_t = typename sub< v1, v2 >::type
- template<typename v1 , typename v2 >
  using mul_t = typename mul< v1, v2 >::type
- template<typename v1 , typename v2 >
  using div_t = typename div< v1, v2 >::type

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

  modulus operator yields v1 % v2 for example : i32::mod_t<i32::val<7>, i32::val<2>>
- template<typename v1 , typename v2 >
  using gt_t = typename gt< v1, v2 >::type
- template<typename v1 , typename v2 >
  using lt_t = typename lt< v1, v2 >::type
- template<typename v1 , typename v2 >
  using eq_t = typename eq< v1, v2 >::type
- template<typename v1 , typename v2 >
  using gcd_t = gcd_t< i32, v1, v2 >
- template<typename v >
  using pos_t = typename pos< v >::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
- template<typename v >
  static constexpr bool pos_v = pos_t<v>::value

8.15.1 Detailed Description

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

8.15.2 Member Typedef Documentation

8.15.2.1 add_t

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

8.15.2.2 div_t

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

8.15.2.3 eq_t

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

8.15.2.4 gcd_t

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

8.15.2.5 gt_t

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

8.15.2.6 inject_constant_t

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

8.15.2.7 inject_ring_t

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

8.15.2.8 inner_type

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

8.15.2.9 lt_t

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

8.15.2.10 mod_t

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

modulus operator yields v1 % v2 for example : i32::mod_t<i32::val<7>, i32::val<2>>

Template Parameters

v1	a value in i32
v2	a value in i32

8.15.2.11 mul_t

```
template<typename v1 , typename v2 >
```

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

8.15.2.12 one

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

constant one

8.15.2.13 pos_t

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

8.15.2.14 sub_t

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

8.15.2.15 zero

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

constant zero

8.15.3 Member Data Documentation

8.15.3.1 eq_v

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

8.15.3.2 is_euclidean_domain

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

integers are an euclidean domain

8.15.3.3 is_field

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

integers are not a field

8.15.3.4 pos_v

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

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

- src/aerobus.h

8.16 aerobus::i64 Struct Reference

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

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

Classes

- struct val

  values in i64

Public Types

- using inner_type = int64_t

  type of represented values
- template<auto x>
  using inject_constant_t = val< static_cast< int64_t >(x)>
- 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
- template<typename v1 , typename v2 >
  using sub_t = typename sub< v1, v2 >::type
- template<typename v1 , typename v2 >
  using mul_t = typename mul< v1, v2 >::type
- template<typename v1 , typename v2 >
  using div_t = typename div< v1, v2 >::type
- template<typename v1 , typename v2 >
  using mod_t = typename remainder< v1, v2 >::type
- template<typename v1 , typename v2 >
  using gt_t = typename gt< v1, v2 >::type
- template<typename v1 , typename v2 >
  using lt_t = typename lt< v1, v2 >::type
- template<typename v1 , typename v2 >
  using eq_t = typename eq< v1, v2 >::type
- template<typename v1 , typename v2 >
  using gcd_t = gcd_t< i64, v1, v2 >
- template<typename v >
  using pos_t = typename pos< v >::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
- template<typename v1 , typename v2 >
  static constexpr bool eq_v = eq_t<v1, v2>::value
- template<typename v >
  static constexpr bool pos_v = pos_t<v>::value

8.16.1 Detailed Description

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

8.16.2 Member Typedef Documentation

8.16.2.1 add_t

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

8.16.2.2 div_t

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

8.16.2.3 eq_t

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

8.16.2.4 gcd_t

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

8.16.2.5 gt_t

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

8.16.2.6 inject_constant_t

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

8.16.2.7 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

8.16.2.8 inner_type

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

type of represented values

8.16.2.9 lt_t

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

8.16.2.10 mod_t

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

8.16.2.11 mul_t

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

8.16.2.12 one

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

constant one

8.16.2.13 pos_t

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

8.16.2.14 sub_t

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

8.16.2.15 zero

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

constant zero

8.16.3 Member Data Documentation

8.16.3.1 eq_v

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

8.16.3.2 gt_v

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

strictly greater operator yields v1 > v2 as boolean value

Template Parameters

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

8.16.3.3 is_euclidean_domain

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

integers are an euclidean domain

8.16.3.4 is_field

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

integers are not a field

8.16.3.5 lt_v

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

8.16.3.6 pos_v

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

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

- src/aerobus.h

8.17 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

8.17.1 Detailed Description

template<size_t n>
struct aerobus::is_prime< n >

checks if n is prime

Template Parameters

n	

8.17.2 Member Data Documentation

8.17.2.1 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

8.18 aerobus::polynomial< Ring > Struct Template Reference

`#include <aerobus.h>`

Classes

- struct val
  values (seen as types) in polynomial ring
- struct val< coeffN >
  specialization for constants

Public Types

- using zero = val< typename Ring::zero >
  constant zero
- using one = val< typename Ring::one >
  constant one
- using X = val< typename Ring::one, typename Ring::zero >
  generator
- template<typename P >
  using simplify_t = typename simplify< P >::type
  simplifies a polynomial (recursively deletes highest degree if zero, do nothing otherwise)
- template<typename v1 , typename v2 >
  using add_t = typename add< v1, v2 >::type
  adds two polynomials
- template<typename v1 , typename v2 >
  using sub_t = typename sub< v1, v2 >::type
  substraction of two polynomials
- template<typename v1 , typename v2 >
  using mul_t = typename mul< v1, v2 >::type
  multiplication of two polynomials
- template<typename v1 , typename v2 >
  using eq_t = typename eq_helper< v1, v2 >::type
  equality operator
- template<typename v1 , typename v2 >
  using lt_t = typename lt_helper< v1, v2 >::type
  strict less operator
- template<typename v1 , typename v2 >
  using gt_t = typename gt_helper< v1, v2 >::type
  strict greater operator
- template<typename v1 , typename v2 >
  using div_t = typename div< v1, v2 >::q_type
  division operator

- template<typename v1 , typename v2 >
  using mod_t = typename div_helper< v1, v2, zero, v1 >::mod_type

  modulo operator
- template<typename coeff , size_t deg>
  using monomial_t = typename monomial< coeff, deg >::type

  monomial : coeff X^ 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 > >
- template<typename v >
  using inject_ring_t = val< v >

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

8.18.1 Detailed Description

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

polynomial with coefficients in Ring Ring must be an integral domain

8.18.2 Member Typedef Documentation

8.18.2.1 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	

8.18.2.2 derive_t

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

derivation operator

Template Parameters

v	

8.18.2.3 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	

8.18.2.4 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	

8.18.2.5 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	

8.18.2.6 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	

8.18.2.7 inject_constant_t

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

8.18.2.8 inject_ring_t

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

8.18.2.9 lt_t

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

strict less operator

Template Parameters

v1	
v2	

8.18.2.10 mod_t

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

modulo operator

Template Parameters

v1	
v2	

8.18.2.11 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	

8.18.2.12 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	

8.18.2.13 one

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

constant one

8.18.2.14 pos_t

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

checks for positivity (an > 0)

Template Parameters

v	

8.18.2.15 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

P	

8.18.2.16 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	

8.18.2.17 X

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

generator

8.18.2.18 zero

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

constant zero

8.18.3 Member Data Documentation

8.18.3.1 is_euclidean_domain

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

8.18.3.2 is_field

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

8.18.3.3 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

8.19 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

8.19.1 Detailed Description

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

removes types from head of the list

8.19.2 Member Typedef Documentation

8.19.2.1 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

8.19.2.2 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

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

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

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

Classes

- struct val
  projection values in the quotient ring

Public Types

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

modulus operator

- template<typename v1 , typename v2 >
  using eq_t = typename Ring::template eq_t< typename v1::type, typename v2::type >

  equality operator (as type)
- template<typename v1 >
  using pos_t = std::true_type

  positivity operator always true
- template<auto x>
  using inject_constant_t = val< typename Ring::template inject_constant_t< x > >
- template<typename v >
  using inject_ring_t = val< v >

Static Public Attributes

- template<typename v1 , typename v2 >
  static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value

  addition operator (as boolean value)
- template<typename v >
  static constexpr bool pos_v = pos_t<v>::value

  positivity operator always true
- static constexpr bool is_euclidean_domain = true

  quotien rings are euclidean domain

8.20.1 Detailed Description

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

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

Template Parameters

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

8.20.2 Member Typedef Documentation

8.20.2.1 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

8.20.2.2 div_t

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

division operator

Template Parameters

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

8.20.2.3 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

8.20.2.4 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>
>
```

8.20.2.5 inject_ring_t

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

8.20.2.6 mod_t

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

modulus operator

Template Parameters

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

8.20.2.7 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::type,
typename v2::type> >
```

substraction operator

Template Parameters

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

8.20.2.8 one

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

one

8.20.2.9 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

8.20.2.10 zero

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

zero value

8.20.3 Member Data Documentation

8.20.3.1 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

8.20.3.2 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

8.20.3.3 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

8.21 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

8.21.1 Detailed Description

template<typename... Ts>
template<size_t index>
struct aerobus::type_list< Ts >::split< index >

splits list at index

Template Parameters

index	

8.21.2 Member Typedef Documentation

8.21.2.1 head

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

8.21.2.2 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

8.22 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

8.22.1 Detailed Description

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

Empty pure template struct to handle type list.

8.22.2 Member Typedef Documentation

8.22.2.1 at

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

returns type at index

Template Parameters

index	

8.22.2.2 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

U	

8.22.2.3 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	

8.22.2.4 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	

8.22.2.5 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

T	

8.22.2.6 remove

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

removes type at index

Template Parameters

index	

8.22.3 Member Data Documentation

8.22.3.1 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

8.23 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

8.23.1 Detailed Description

specialization for empty type list

8.23.2 Member Typedef Documentation

8.23.2.1 concat

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

8.23.2.2 insert

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

8.23.2.3 push_back

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

8.23.2.4 push_front

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

8.23.3 Member Data Documentation

8.23.3.1 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

8.24 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

8.24.1 Detailed Description

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

values in i32, again represented as types

Template Parameters

x	an actual integer

8.24.2 Member Typedef Documentation

8.24.2.1 enclosing_type

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

Enclosing ring type.

8.24.2.2 is_zero_t

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

is value zero

8.24.3 Member Function Documentation

8.24.3.1 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

8.24.3.2 to_string()

```
template<int32_t x>
static std::string aerobus::i32::val< x >::to_string ( )  [inline], [static]
```

string representation of value

8.24.4 Member Data Documentation

8.24.4.1 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

8.25 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

8.25.1 Detailed Description

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

values in i64

Template Parameters

x	an actual integer

8.25.2 Member Typedef Documentation

8.25.2.1 enclosing_type

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

enclosing ring type

8.25.2.2 inner_type

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

type of represented values

8.25.2.3 is_zero_t

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

is value zero

8.25.3 Member Function Documentation

8.25.3.1 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)

8.25.3.2 to_string()

```
template<int64_t x>
static std::string aerobus::i64::val< x >::to_string ( )  [inline], [static]
```

string representation

8.25.4 Member Data Documentation

8.25.4.1 v

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

actual value

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

- src/aerobus.h

8.26 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

Static Public Member Functions

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

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

8.26.1 Detailed Description

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

values (seen as types) in polynomial ring

Template Parameters

coeffN	high degree coefficient
...coeffs	lower degree coefficients

8.26.2 Member Typedef Documentation

8.26.2.1 aN

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

heavy weight coefficient (non zero)

8.26.2.2 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	

8.26.2.3 enclosing_type

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

enclosing ring type

8.26.2.4 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

8.26.2.5 ring_type

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

ring coefficients live in

8.26.2.6 strip

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

remove largest coefficient

8.26.3 Member Function Documentation

8.26.3.1 eval()

```
template<typename Ring >
template<typename coeffN , typename...  coeffs>
template<typename valueRing >
static constexpr DEVICE INLINED valueRing aerobus::polynomial< Ring >::val< coeffN, coeffs
>::eval (
            const valueRing & x )  [inline], [static], [constexpr]
```

evaluates polynomial seen as a function operating on ValueRing

Template Parameters

valueRing	usually float or double

Parameters

x	value

Returns

P(x)

8.26.3.2 to_string()

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

get a string representation of polynomial

Returns

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

8.26.4 Member Data Documentation

8.26.4.1 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

8.26.4.2 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↩
::value  [static], [constexpr]
```

true if polynomial is constant zero

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

- src/aerobus.h

8.27 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 > >

8.27.1 Detailed Description

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

projection values in the quotient ring

Template Parameters

V	a value from 'Ring'

8.27.2 Member Typedef Documentation

8.27.2.1 raw_t

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

8.27.2.2 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

8.28 aerobus::zpz< p >::val< x > Struct Template Reference

values in zpz

`#include <aerobus.h>`

Public Types

- using enclosing_type = zpz< p >
  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

8.28.1 Detailed Description

template<int32_t p>
template<int32_t x>
struct aerobus::zpz< p >::val< x >

values in zpz

Template Parameters

x	an integer

8.28.2 Member Typedef Documentation

8.28.2.1 enclosing_type

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

enclosing ring type

8.28.2.2 is_zero_t

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

true_type if zero

8.28.3 Member Function Documentation

8.28.3.1 get()

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

get value as valueType

Template Parameters

valueType	an arithmetic type, such as float

8.28.3.2 to_string()

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

string representation

Returns

a string representation

8.28.4 Member Data Documentation

8.28.4.1 is_zero_v

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

true if zero

8.28.4.2 v

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

actual value

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

- src/aerobus.h

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

specialization for constants

`#include <aerobus.h>`

Classes

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

Public Types

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

Static Public Member Functions

- static std::string to_string ()

Static Public Attributes

- static constexpr size_t degree = 0

  degree
- static constexpr bool is_zero_v = is_zero_t::value

8.29.1 Detailed Description

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

specialization for constants

Template Parameters

coeffN	

8.29.2 Member Typedef Documentation

8.29.2.1 aN

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

8.29.2.2 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
```

8.29.2.3 enclosing_type

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

enclosing ring type

8.29.2.4 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>
```

8.29.2.5 ring_type

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

ring coefficients live in

8.29.2.6 strip

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

8.29.3 Member Function Documentation

8.29.3.1 to_string()

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

8.29.4 Member Data Documentation

8.29.4.1 degree

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

degree

8.29.4.2 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

8.30 aerobus::zpz< p > 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 lt_t = typename lt< v1, v2 >::type
  strictly smaller operator (type)
- template<typename v1 , typename v2 >
  using eq_t = typename eq< v1, v2 >::type
  equality operator (type)
- template<typename v1 , typename v2 >
  using gcd_t = gcd_t< i32, v1, v2 >
  greatest common divisor
- template<typename v1 >
  using pos_t = typename pos< v1 >::type
  positivity operator (type)

Static Public Attributes

- static constexpr bool is_field = is_prime<p>::value
  true iff p is prime
- static constexpr bool is_euclidean_domain = true
  always true
- template<typename v1 , typename v2 >
  static constexpr bool gt_v = gt_t<v1, v2>::value
  strictly greater operator (booleanvalue)
- template<typename v1 , typename v2 >
  static constexpr bool lt_v = lt_t<v1, v2>::value
  strictly smaller operator (booleanvalue)
- template<typename v1 , typename v2 >
  static constexpr bool eq_v = eq_t<v1, v2>::value
  equality operator (booleanvalue)
- template<typename v >
  static constexpr bool pos_v = pos_t<v>::value
  positivity operator (boolean value)

8.30.1 Detailed Description

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

congruence classes of integers modulo p (32 bits)

if p is prime, zpz

is a field

Template Parameters

p	a integer

8.30.2 Member Typedef Documentation

8.30.2.1 add_t

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

addition operator

Template Parameters

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

8.30.2.2 div_t

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

division operator

Template Parameters

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

8.30.2.3 eq_t

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

equality operator (type)

Template Parameters

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

8.30.2.4 gcd_t

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

greatest common divisor

Template Parameters

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

8.30.2.5 gt_t

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

strictly greater operator (type)

Template Parameters

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

8.30.2.6 inject_constant_t

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

injects a constant integer into zpz

Template Parameters

x	an integer

8.30.2.7 inner_type

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

underlying type for values

8.30.2.8 lt_t

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

8.30.2.9 mod_t

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

modulo operator

Template Parameters

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

8.30.2.10 mul_t

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

multiplication operator

Template Parameters

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

8.30.2.11 one

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

one value

8.30.2.12 pos_t

```
template<int32_t p>
template<typename v1 >
using aerobus::zpz< p >::pos_t = typename pos<v1>::type
```

positivity operator (type)

Template Parameters

v1	a value in zpz::val

8.30.2.13 sub_t

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

substraction operator

Template Parameters

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

8.30.2.14 zero

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

zero value

8.30.3 Member Data Documentation

8.30.3.1 eq_v

```
template<int32_t p>
template<typename v1 , typename v2 >
constexpr bool aerobus::zpz< p >::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

8.30.3.2 gt_v

```
template<int32_t p>
template<typename v1 , typename v2 >
constexpr bool aerobus::zpz< p >::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

8.30.3.3 is_euclidean_domain

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

always true

8.30.3.4 is_field

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

true iff p is prime

8.30.3.5 lt_v

```
template<int32_t p>
template<typename v1 , typename v2 >
constexpr bool aerobus::zpz< p >::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

8.30.3.6 pos_v

```
template<int32_t p>
template<typename v >
constexpr bool aerobus::zpz< p >::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
00054         return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
00055         #else
00056         return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
00057         #endif
00058     }
00059 }  // namespace aerobus
00060
00061 // concepts
00062 namespace aerobus {
00064     template <typename R>
00065     concept IsRing = requires {
00066         typename R::one;
00067         typename R::zero;
00068         typename R::template add_t<typename R::one, typename R::one>;
00069         typename R::template sub_t<typename R::one, typename R::one>;
00070         typename R::template mul_t<typename R::one, typename R::one>;
00071     };
00072
00074     template <typename R>
00075     concept IsEuclideanDomain = IsRing<R> && requires {
00076         typename R::template div_t<typename R::one, typename R::one>;
00077         typename R::template mod_t<typename R::one, typename R::one>;
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>;
00084         R::is_euclidean_domain == true;
00085     };
00086
00088     template<typename R>
00089     concept IsField = IsEuclideanDomain<R> && requires {
00090         R::is_field == true;
00091     };
00092 }  // namespace aerobus
00093
00094 #ifdef WITH_CUDA_FP16
00095 // all this shit is required because of NVIDIA bug https://developer.nvidia.com/bugs/4863696
00096 namespace aerobus {
00097     namespace internal {
00098         static consteval DEVICE uint16_t my_internal_float2half(
00099             const float f, uint32_t &sign, uint32_t &remainder) {
00100             uint32_t x;
00101             uint32_t u;
00102             uint32_t result;
00103             x = std::bit_cast<int32_t>(f);
00104             u = (x & 0x7fffffffU);
00105             sign = ((x >> 16U) & 0x8000U);
00106             // NaN/+Inf/-Inf
00107             if (u >= 0x7f800000U) {
00108                 remainder = 0U;
00109                 result = ((u == 0x7f800000U) ? (sign | 0x7c00U) : 0x7fffU);
00110             } else if (u > 0x477fefffU) {  // Overflows
00111                 remainder = 0x80000000U;
00112                 result = (sign | 0x7bffU);
00113             } else if (u >= 0x38800000U) {  // Normal numbers
00114                 remainder = u << 19U;
00115                 u -= 0x38000000U;
00116                 result = (sign | (u >> 13U));
00117             } else if (u < 0x33000001U) {  // +0/-0
00118                 remainder = u;
00119                 result = sign;
00120             } else {  // Denormal numbers
00121                 const uint32_t exponent = u >> 23U;
00122                 const uint32_t shift = 0x7eU - exponent;
00123                 uint32_t mantissa = (u & 0x7fffffU);
00124                 mantissa |= 0x800000U;
00125                 remainder = mantissa << (32U - shift);
00126                 result = (sign | (mantissa >> shift));
00127                 result &= 0x0000FFFFU;
```

```
00128             }
00129             return static_cast<uint16_t>(result);
00130         }
00131 
00132         static consteval DEVICE __half my_float2half_rn(const float a) {
00133             __half val;
00134             __half_raw r;
00135             uint32_t sign = 0U;
00136             uint32_t remainder = 0U;
00137             r.x = my_internal_float2half(a, sign, remainder);
00138             if ((remainder > 0x80000000U) || ((remainder == 0x80000000U) && ((r.x & 0x1U) != 0U))) {
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>
00162         struct int16_convert_helper<__half, x> {
00163             static constexpr __half value() {
00164                 return convert_int16_to_half<x>;
00165             }
00166         };
00167 
00168         template <int16_t x>
00169         struct int16_convert_helper<__half2, x> {
00170             static constexpr __half2 value() {
00171                 return __half2(convert_int16_to_half<x>, convert_int16_to_half<x>);
00172             }
00173         };
00174     }  // namespace internal
00175 }  // namespace aerobus
00176 #endif
00177 
00178 //  cast
00179 namespace aerobus {
00180     namespace internal {
00181         template<typename Out, typename In>
00182         struct staticcast {
00183             template<auto x>
00184             static consteval INLINED DEVICE Out func() {
00185                 return static_cast<Out>(x);
00186             }
00187         };
00188 
00189         #ifdef WITH_CUDA_FP16
00190         template<>
00191         struct staticcast<__half, int16_t> {
00192             template<int16_t x>
00193             static consteval INLINED DEVICE __half func() {
00194                 return int16_convert_helper<__half, x>::value();
00195             }
00196         };
00197 
00198         template<>
00199         struct staticcast<__half2, int16_t> {
00200             template<int16_t x>
00201             static consteval INLINED DEVICE __half2 func() {
00202                 return int16_convert_helper<__half2, x>::value();
00203             }
00204         };
00205         #endif
00206     }  // namespace internal
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 * y + z;
00219             }
00220         };
00221 
00222         template<>
00223         struct fma_helper<float> {
00224             static constexpr INLINED DEVICE float eval(const float x, const float y, const float z) {
00225                 return x * y + z;
00226             }
00227         };
00228 
00229         template<>
00230         struct fma_helper<int32_t> {
00231             static constexpr INLINED DEVICE int16_t eval(const int16_t x, const int16_t y, const
      int16_t z) {
00232                 return x * y + z;
00233             }
00234         };
00235 
00236         template<>
00237         struct fma_helper<int16_t> {
00238             static constexpr INLINED DEVICE int32_t eval(const int32_t x, const int32_t y, const
      int32_t z) {
00239                 return x * y + z;
00240             }
00241         };
00242 
00243         template<>
00244         struct fma_helper<int64_t> {
00245             static constexpr INLINED DEVICE int64_t eval(const int64_t x, const int64_t y, const
      int64_t z) {
00246                 return x * y + z;
00247             }
00248         };
00249 
00250         #ifdef WITH_CUDA_FP16
00251         template<>
00252         struct fma_helper<__half> {
00253             static constexpr INLINED DEVICE __half eval(const __half x, const __half y, const __half
      z) {
00254                 #ifdef __CUDA_ARCH__
00255                 return __hfma(x, y, z);
00256                 #else
00257                 return x * y + z;
00258                 #endif
00259             }
00260         };
00261         template<>
00262         struct fma_helper<__half2> {
00263             static constexpr INLINED DEVICE __half2 eval(const __half2 x, const __half2 y, const
      __half2 z) {
00264                 #ifdef __CUDA_ARCH__
00265                 return __hfma2(x, y, z);
00266                 #else
00267                 return x * y + z;
00268                 #endif
00269             }
00270         };
00271         #endif
00272     }  // namespace internal
00273 }  // namespace aerobus
00274 
00275 // utilities
00276 namespace aerobus {
00277     namespace internal {
00278         template<template<typename...> typename TT, typename T>
00279         struct is_instantiation_of : std::false_type { };
00280 
00281         template<template<typename...> typename TT, typename... Ts>
00282         struct is_instantiation_of<TT, TT<Ts...>> : std::true_type { };
00283 
00284         template<template<typename...> typename TT, typename T>
00285         inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00286 
00287         template <int64_t i, typename T, typename... Ts>
00288         struct type_at {
00289             static_assert(i < sizeof...(Ts) + 1, "index out of range");
00290             using type = typename type_at<i - 1, Ts...>::type;
00291         };
00292 
00293         template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00294             using type = T;
00295         };
```

```
00296
00297         template <size_t i, typename... Ts>
00298         using type_at_t = typename type_at<i, Ts...>::type;
00299
00300
00301         template<size_t n, size_t i, typename E = void>
00302         struct _is_prime {};
00303
00304         template<size_t i>
00305         struct _is_prime<0, i> {
00306             static constexpr bool value = false;
00307         };
00308
00309         template<size_t i>
00310         struct _is_prime<1, i> {
00311             static constexpr bool value = false;
00312         };
00313
00314         template<size_t i>
00315         struct _is_prime<2, i> {
00316             static constexpr bool value = true;
00317         };
00318
00319         template<size_t i>
00320         struct _is_prime<3, i> {
00321             static constexpr bool value = true;
00322         };
00323
00324         template<size_t i>
00325         struct _is_prime<5, i> {
00326             static constexpr bool value = true;
00327         };
00328
00329         template<size_t i>
00330         struct _is_prime<7, i> {
00331             static constexpr bool value = true;
00332         };
00333
00334         template<size_t n, size_t i>
00335         struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)>> {
00336             static constexpr bool value = false;
00337         };
00338
00339         template<size_t n, size_t i>
00340         struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)>> {
00341             static constexpr bool value = false;
00342         };
00343
00344         template<size_t n, size_t i>
00345         struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)>> {
00346             static constexpr bool value = true;
00347         };
00348
00349         template<size_t n, size_t i>
00350         struct _is_prime<n, i, std::enable_if_t<(
00351             n % i == 0 &&
00352             n >= 9 &&
00353             n % 3 != 0 &&
00354             n % 2 != 0 &&
00355             i * i > n)>> {
00356             static constexpr bool value = true;
00357         };
00358
00359         template<size_t n, size_t i>
00360         struct _is_prime<n, i, std::enable_if_t<(
00361             n % (i+2) == 0 &&
00362             n >= 9 &&
00363             n % 3 != 0 &&
00364             n % 2 != 0 &&
00365             i * i <= n)>> {
00366             static constexpr bool value = true;
00367         };
00368
00369         template<size_t n, size_t i>
00370         struct _is_prime<n, i, std::enable_if_t<(
00371                 n % (i+2) != 0 &&
00372                 n % i != 0 &&
00373                 n >= 9 &&
00374                 n % 3 != 0 &&
00375                 n % 2 != 0 &&
00376                 (i * i <= n))>> {
00377             static constexpr bool value = _is_prime<n, i+6>::value;
00378         };
00379
00380     }  // namespace internal
00381
00384     template<size_t n>
```

```
00385     struct is_prime {
00387         static constexpr bool value = internal::_is_prime<n, 5>::value;
00388     };
00389
00393     template<size_t n>
00394     static constexpr bool is_prime_v = is_prime<n>::value;
00395
00396     // gcd
00397     namespace internal {
00398         template <std::size_t... Is>
00399         constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
00400             -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
00401
00402         template <std::size_t N>
00403         using make_index_sequence_reverse
00404             = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00405
00411         template<typename Ring, typename E = void>
00412         struct gcd;
00413
00414         template<typename Ring>
00415         struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {
00416             template<typename A, typename B, typename E = void>
00417             struct gcd_helper {};
00418
00419             // B = 0, A > 0
00420             template<typename A, typename B>
00421             struct gcd_helper<A, B, std::enable_if_t<
00422                 ((B::is_zero_t::value) &&
00423                     (Ring::template gt_t<A, typename Ring::zero>::value))» {
00424                 using type = A;
00425             };
00426
00427             // B = 0, A < 0
00428             template<typename A, typename B>
00429             struct gcd_helper<A, B, std::enable_if_t<
00430                 ((B::is_zero_t::value) &&
00431                     !(Ring::template gt_t<A, typename Ring::zero>::value))» {
00432                 using type = typename Ring::template sub_t<typename Ring::zero, A>;
00433             };
00434
00435             // B != 0
00436             template<typename A, typename B>
00437             struct gcd_helper<A, B, std::enable_if_t<
00438                 (!B::is_zero_t::value)
00439                 » {
00440             private: // NOLINT
00441                 // A / B
00442                 using k = typename Ring::template div_t<A, B>;
00443                 // A - (A/B)*B = A % B
00444                 using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
00445
00446             public:
00447                 using type = typename gcd_helper<B, m>::type;
00448             };
00449
00450             template<typename A, typename B>
00451             using type = typename gcd_helper<A, B>::type;
00452         };
00453     }  // namespace internal
00454
00455     // vadd and vmul
00456     namespace internal {
00457         template<typename... vals>
00458         struct vmul {};
00459
00460         template<typename v1, typename... vals>
00461         struct vmul<v1, vals...> {
00462             using type = typename v1::enclosing_type::template mul_t<v1, typename
      vmul<vals...>::type>;
00463         };
00464
00465         template<typename v1>
00466         struct vmul<v1> {
00467             using type = v1;
00468         };
00469
00470         template<typename... vals>
00471         struct vadd {};
00472
00473         template<typename v1, typename... vals>
00474         struct vadd<v1, vals...> {
00475             using type = typename v1::enclosing_type::template add_t<v1, typename
      vadd<vals...>::type>;
00476         };
00477
00478         template<typename v1>
```

```
00479         struct vadd<v1> {
00480             using type = v1;
00481         };
00482     }  // namespace internal
00483
00486     template<typename T, typename A, typename B>
00487     using gcd_t = typename internal::gcd<T>::template type<A, B>;
00488
00492     template<typename... vals>
00493     using vadd_t = typename internal::vadd<vals...>::type;
00494
00498     template<typename... vals>
00499     using vmul_t = typename internal::vmul<vals...>::type;
00500
00504     template<typename val>
00505     requires IsEuclideanDomain<typename val::enclosing_type>
00506     using abs_t = std::conditional_t<
00507                     val::enclosing_type::template pos_v<val>,
00508                     val, typename val::enclosing_type::template
      sub_t<typename val::enclosing_type::zero, val>>;
00509 }  // namespace aerobus
00510
00511 // embedding
00512 namespace aerobus {
00517     template<typename Small, typename Large, typename E = void>
00518     struct Embed;
00519 }  // namespace aerobus
00520
00521 namespace aerobus {
00526     template<typename Ring, typename X>
00527     requires IsRing<Ring>
00528     struct Quotient {
00531         template <typename V>
00532         struct val {
00533          public:
00534             using raw_t = V;
00535             using type = abs_t<typename Ring::template mod_t<V, X>>;
00536         };
00537
00539         using zero = val<typename Ring::zero>;
00540
00542         using one = val<typename Ring::one>;
00543
00547         template<typename v1, typename v2>
00548         using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00549
00553         template<typename v1, typename v2>
00554         using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00555
00559         template<typename v1, typename v2>
00560         using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00561
00565         template<typename v1, typename v2>
00566         using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00567
00571         template<typename v1, typename v2>
00572         using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00573
00577         template<typename v1, typename v2>
00578         static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00579
00583         template<typename v1>
00584         using pos_t = std::true_type;
00585
00589         template<typename v>
00590         static constexpr bool pos_v = pos_t<v>::value;
00591
00593         static constexpr bool is_euclidean_domain = true;
00594
00600         template<auto x>
00601         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00602
00608         template<typename v>
00609         using inject_ring_t = val<v>;
00610     };
00611
00615     template<typename Ring, typename X>
00616     struct Embed<Quotient<Ring, X>, Ring> {
00619         template<typename val>
00620         using type = typename val::raw_t;
00621     };
00622 }  // namespace aerobus
00623
00624 // type_list
00625 namespace aerobus {
00627     template <typename... Ts>
00628     struct type_list;
```

```
00629
00630     namespace internal {
00631         template <typename T, typename... Us>
00632         struct pop_front_h {
00633             using tail = type_list<Us...>;
00634             using head = T;
00635         };
00636
00637         template <size_t index, typename L1, typename L2>
00638         struct split_h {
00639          private:
00640             static_assert(index <= L2::length, "index ouf of bounds");
00641             using a = typename L2::pop_front::type;
00642             using b = typename L2::pop_front::tail;
00643             using c = typename L1::template push_back<a>;
00644
00645          public:
00646             using head = typename split_h<index - 1, c, b>::head;
00647             using tail = typename split_h<index - 1, c, b>::tail;
00648         };
00649
00650         template <typename L1, typename L2>
00651         struct split_h<0, L1, L2> {
00652             using head = L1;
00653             using tail = L2;
00654         };
00655
00656         template <size_t index, typename L, typename T>
00657         struct insert_h {
00658             static_assert(index <= L::length, "index ouf of bounds");
00659             using s = typename L::template split<index>;
00660             using left = typename s::head;
00661             using right = typename s::tail;
00662             using ll = typename left::template push_back<T>;
00663             using type = typename ll::template concat<right>;
00664         };
00665
00666         template <size_t index, typename L>
00667         struct remove_h {
00668             using s = typename L::template split<index>;
00669             using left = typename s::head;
00670             using right = typename s::tail;
00671             using rr = typename right::pop_front::tail;
00672             using type = typename left::template concat<rr>;
00673         };
00674     }  // namespace internal
00675
00679     template <typename... Ts>
00680     struct type_list {
00681      private:
00682         template <typename T>
00683         struct concat_h;
00684
00685         template <typename... Us>
00686         struct concat_h<type_list<Us...>> {
00687             using type = type_list<Ts..., Us...>;
00688         };
00689
00690      public:
00692         static constexpr size_t length = sizeof...(Ts);
00693
00696         template <typename T>
00697         using push_front = type_list<T, Ts...>;
00698
00701         template <size_t index>
00702         using at = internal::type_at_t<index, Ts...>;
00703
00705         struct pop_front {
00707             using type = typename internal::pop_front_h<Ts...>::head;
00709             using tail = typename internal::pop_front_h<Ts...>::tail;
00710         };
00711
00714         template <typename T>
00715         using push_back = type_list<Ts..., T>;
00716
00719         template <typename U>
00720         using concat = typename concat_h<U>::type;
00721
00724         template <size_t index>
00725         struct split {
00726          private:
00727             using inner = internal::split_h<index, type_list<>, type_list<Ts...>>;
00728
00729          public:
00730             using head = typename inner::head;
00731             using tail = typename inner::tail;
00732         };
```

```
00733
00737         template <typename T, size_t index>
00738         using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00739
00742         template <size_t index>
00743         using remove = typename internal::remove_h<index, type_list<Ts...»::type;
00744     };
00745
00747     template <>
00748     struct type_list<> {
00749         static constexpr size_t length = 0;
00750
00751         template <typename T>
00752         using push_front = type_list<T>;
00753
00754         template <typename T>
00755         using push_back = type_list<T>;
00756
00757         template <typename U>
00758         using concat = U;
00759
00760         // TODO(jewave): assert index == 0
00761         template <typename T, size_t index>
00762         using insert = type_list<T>;
00763     };
00764 }  // namespace aerobus
00765
00766 // i16
00767 #ifdef WITH_CUDA_FP16
00768 // i16
00769 namespace aerobus {
00771     struct i16 {
00772         using inner_type = int16_t;
00775         template<int16_t x>
00776         struct val {
00778             using enclosing_type = i16;
00780             static constexpr int16_t v = x;
00781
00784             template<typename valueType>
00785             static constexpr INLINED DEVICE valueType get() {
00786                 return internal::template int16_convert_helper<valueType, x>::value();
00787             }
00788
00790             using is_zero_t = std::bool_constant<x == 0>;
00791
00793             static std::string to_string() {
00794                 return std::to_string(x);
00795             }
00796         };
00797
00799         using zero = val<0>;
00801         using one = val<1>;
00803         static constexpr bool is_field = false;
00805         static constexpr bool is_euclidean_domain = true;
00809         template<auto x>
00810         using inject_constant_t = val<static_cast<int16_t>(x)>;
00811
00812         template<typename v>
00813         using inject_ring_t = v;
00814
00815      private:
00816         template<typename v1, typename v2>
00817         struct add {
00818             using type = val<v1::v + v2::v>;
00819         };
00820
00821         template<typename v1, typename v2>
00822         struct sub {
00823             using type = val<v1::v - v2::v>;
00824         };
00825
00826         template<typename v1, typename v2>
00827         struct mul {
00828             using type = val<v1::v* v2::v>;
00829         };
00830
00831         template<typename v1, typename v2>
00832         struct div {
00833             using type = val<v1::v / v2::v>;
00834         };
00835
00836         template<typename v1, typename v2>
00837         struct remainder {
00838             using type = val<v1::v % v2::v>;
00839         };
00840
00841         template<typename v1, typename v2>
```

```
00842         struct gt {
00843             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00844         };
00845 
00846         template<typename v1, typename v2>
00847         struct lt {
00848             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00849         };
00850 
00851         template<typename v1, typename v2>
00852         struct eq {
00853             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00854         };
00855 
00856         template<typename v1>
00857         struct pos {
00858             using type = std::bool_constant<(v1::v > 0)>;
00859         };
00860 
00861     public:
00867         template<typename v1, typename v2>
00868         using add_t = typename add<v1, v2>::type;
00869 
00875         template<typename v1, typename v2>
00876         using sub_t = typename sub<v1, v2>::type;
00877 
00883         template<typename v1, typename v2>
00884         using mul_t = typename mul<v1, v2>::type;
00885 
00891         template<typename v1, typename v2>
00892         using div_t = typename div<v1, v2>::type;
00893 
00899         template<typename v1, typename v2>
00900         using mod_t = typename remainder<v1, v2>::type;
00901 
00907         template<typename v1, typename v2>
00908         using gt_t = typename gt<v1, v2>::type;
00909 
00915         template<typename v1, typename v2>
00916         using lt_t = typename lt<v1, v2>::type;
00917 
00923         template<typename v1, typename v2>
00924         using eq_t = typename eq<v1, v2>::type;
00925 
00930         template<typename v1, typename v2>
00931         static constexpr bool eq_v = eq_t<v1, v2>::value;
00932 
00938         template<typename v1, typename v2>
00939         using gcd_t = gcd_t<i16, v1, v2>;
00940 
00945         template<typename v>
00946         using pos_t = typename pos<v>::type;
00947 
00952         template<typename v>
00953         static constexpr bool pos_v = pos_t<v>::value;
00954     };
00955 }  // namespace aerobus
00956 #endif
00957 
00958 // i32
00959 namespace aerobus {
00961     struct i32 {
00962         using inner_type = int32_t;
00965         template<int32_t x>
00966         struct val {
00968             using enclosing_type = i32;
00970             static constexpr int32_t v = x;
00971 
00974             template<typename valueType>
00975             static constexpr DEVICE valueType get() {
00976                 return static_cast<valueType>(x);
00977             }
00978 
00980             using is_zero_t = std::bool_constant<x == 0>;
00981 
00983             static std::string to_string() {
00984                 return std::to_string(x);
00985             }
00986         };
00987 
00989         using zero = val<0>;
00991         using one = val<1>;
00993         static constexpr bool is_field = false;
00995         static constexpr bool is_euclidean_domain = true;
00999         template<auto x>
01000         using inject_constant_t = val<static_cast<int32_t>(x)>;
01001 
```

```
01002         template<typename v>
01003         using inject_ring_t = v;
01004
01005       private:
01006         template<typename v1, typename v2>
01007         struct add {
01008             using type = val<v1::v + v2::v>;
01009         };
01010
01011         template<typename v1, typename v2>
01012         struct sub {
01013             using type = val<v1::v - v2::v>;
01014         };
01015
01016         template<typename v1, typename v2>
01017         struct mul {
01018             using type = val<v1::v* v2::v>;
01019         };
01020
01021         template<typename v1, typename v2>
01022         struct div {
01023             using type = val<v1::v / v2::v>;
01024         };
01025
01026         template<typename v1, typename v2>
01027         struct remainder {
01028             using type = val<v1::v % v2::v>;
01029         };
01030
01031         template<typename v1, typename v2>
01032         struct gt {
01033             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01034         };
01035
01036         template<typename v1, typename v2>
01037         struct lt {
01038             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01039         };
01040
01041         template<typename v1, typename v2>
01042         struct eq {
01043             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01044         };
01045
01046         template<typename v1>
01047         struct pos {
01048             using type = std::bool_constant<(v1::v > 0)>;
01049         };
01050
01051       public:
01057         template<typename v1, typename v2>
01058         using add_t = typename add<v1, v2>::type;
01059
01065         template<typename v1, typename v2>
01066         using sub_t = typename sub<v1, v2>::type;
01067
01073         template<typename v1, typename v2>
01074         using mul_t = typename mul<v1, v2>::type;
01075
01081         template<typename v1, typename v2>
01082         using div_t = typename div<v1, v2>::type;
01083
01089         template<typename v1, typename v2>
01090         using mod_t = typename remainder<v1, v2>::type;
01091
01097         template<typename v1, typename v2>
01098         using gt_t = typename gt<v1, v2>::type;
01099
01105         template<typename v1, typename v2>
01106         using lt_t = typename lt<v1, v2>::type;
01107
01113         template<typename v1, typename v2>
01114         using eq_t = typename eq<v1, v2>::type;
01115
01120         template<typename v1, typename v2>
01121         static constexpr bool eq_v = eq_t<v1, v2>::value;
01122
01128         template<typename v1, typename v2>
01129         using gcd_t = gcd_t<i32, v1, v2>;
01130
01135         template<typename v>
01136         using pos_t = typename pos<v>::type;
01137
01142         template<typename v>
01143         static constexpr bool pos_v = pos_t<v>::value;
01144     };
01145 }  // namespace aerobus
```

```
01146
01147 // i64
01148 namespace aerobus {
01150     struct i64 {
01152         using inner_type = int64_t;
01155         template<int64_t x>
01156         struct val {
01158             using inner_type = int32_t;
01160             using enclosing_type = i64;
01162             static constexpr int64_t v = x;
01163
01166             template<typename valueType>
01167             static constexpr INLINED DEVICE valueType get() {
01168                 return static_cast<valueType>(x);
01169             }
01170
01172             using is_zero_t = std::bool_constant<x == 0>;
01173
01175             static std::string to_string() {
01176                 return std::to_string(x);
01177             }
01178         };
01179
01183         template<auto x>
01184         using inject_constant_t = val<static_cast<int64_t>(x)>;
01185
01190         template<typename v>
01191         using inject_ring_t = v;
01192
01194         using zero = val<0>;
01196         using one = val<1>;
01198         static constexpr bool is_field = false;
01200         static constexpr bool is_euclidean_domain = true;
01201
01202     private:
01203         template<typename v1, typename v2>
01204         struct add {
01205             using type = val<v1::v + v2::v>;
01206         };
01207
01208         template<typename v1, typename v2>
01209         struct sub {
01210             using type = val<v1::v - v2::v>;
01211         };
01212
01213         template<typename v1, typename v2>
01214         struct mul {
01215             using type = val<v1::v* v2::v>;
01216         };
01217
01218         template<typename v1, typename v2>
01219         struct div {
01220             using type = val<v1::v / v2::v>;
01221         };
01222
01223         template<typename v1, typename v2>
01224         struct remainder {
01225             using type = val<v1::v% v2::v>;
01226         };
01227
01228         template<typename v1, typename v2>
01229         struct gt {
01230             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01231         };
01232
01233         template<typename v1, typename v2>
01234         struct lt {
01235             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01236         };
01237
01238         template<typename v1, typename v2>
01239         struct eq {
01240             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01241         };
01242
01243         template<typename v>
01244         struct pos {
01245             using type = std::bool_constant<(v::v > 0)>;
01246         };
01247
01248     public:
01253         template<typename v1, typename v2>
01254         using add_t = typename add<v1, v2>::type;
01255
01260         template<typename v1, typename v2>
01261         using sub_t = typename sub<v1, v2>::type;
01262
```

```
01267         template<typename v1, typename v2>
01268         using mul_t = typename mul<v1, v2>::type;
01269
01275         template<typename v1, typename v2>
01276         using div_t = typename div<v1, v2>::type;
01277
01282         template<typename v1, typename v2>
01283         using mod_t = typename remainder<v1, v2>::type;
01284
01290         template<typename v1, typename v2>
01291         using gt_t = typename gt<v1, v2>::type;
01292
01297         template<typename v1, typename v2>
01298         static constexpr bool gt_v = gt_t<v1, v2>::value;
01299
01305         template<typename v1, typename v2>
01306         using lt_t = typename lt<v1, v2>::type;
01307
01313         template<typename v1, typename v2>
01314         static constexpr bool lt_v = lt_t<v1, v2>::value;
01315
01321         template<typename v1, typename v2>
01322         using eq_t = typename eq<v1, v2>::type;
01323
01329         template<typename v1, typename v2>
01330         static constexpr bool eq_v = eq_t<v1, v2>::value;
01331
01337         template<typename v1, typename v2>
01338         using gcd_t = gcd_t<i64, v1, v2>;
01339
01344         template<typename v>
01345         using pos_t = typename pos<v>::type;
01346
01351         template<typename v>
01352         static constexpr bool pos_v = pos_t<v>::value;
01353     };
01354
01356     template<>
01357     struct Embed<i32, i64> {
01360         template<typename val>
01361         using type = i64::val<static_cast<int64_t>(val::v)>;
01362     };
01363 }  // namespace aerobus
01364
01365 // z/pz
01366 namespace aerobus {
01372     template<int32_t p>
01373     struct zpz {
01375         using inner_type = int32_t;
01376
01379         template<int32_t x>
01380         struct val {
01382             using enclosing_type = zpz<p>;
01384             static constexpr int32_t v = x % p;
01385
01388             template<typename valueType>
01389             static constexpr INLINED DEVICE valueType get() {
01390                 return static_cast<valueType>(x % p);
01391             }
01392
01394             using is_zero_t = std::bool_constant<v == 0>;
01395
01397             static constexpr bool is_zero_v = v == 0;
01398
01401             static std::string to_string() {
01402                 return std::to_string(x % p);
01403             }
01404         };
01405
01408         template<auto x>
01409         using inject_constant_t = val<static_cast<int32_t>(x)>;
01410
01412         using zero = val<0>;
01413
01415         using one = val<1>;
01416
01418         static constexpr bool is_field = is_prime<p>::value;
01419
01421         static constexpr bool is_euclidean_domain = true;
01422
01423      private:
01424         template<typename v1, typename v2>
01425         struct add {
01426             using type = val<(v1::v + v2::v) % p>;
01427         };
01428
01429         template<typename v1, typename v2>
```

```
01430         struct sub {
01431             using type = val<(v1::v - v2::v) % p>;
01432         };
01433 
01434         template<typename v1, typename v2>
01435         struct mul {
01436             using type = val<(v1::v* v2::v) % p>;
01437         };
01438 
01439         template<typename v1, typename v2>
01440         struct div {
01441             using type = val<(v1::v% p) / (v2::v % p)>;
01442         };
01443 
01444         template<typename v1, typename v2>
01445         struct remainder {
01446             using type = val<(v1::v% v2::v) % p>;
01447         };
01448 
01449         template<typename v1, typename v2>
01450         struct gt {
01451             using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01452         };
01453 
01454         template<typename v1, typename v2>
01455         struct lt {
01456             using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01457         };
01458 
01459         template<typename v1, typename v2>
01460         struct eq {
01461             using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01462         };
01463 
01464         template<typename v1>
01465         struct pos {
01466             using type = std::bool_constant<(v1::v > 0)>;
01467         };
01468 
01469      public:
01473         template<typename v1, typename v2>
01474         using add_t = typename add<v1, v2>::type;
01475 
01479         template<typename v1, typename v2>
01480         using sub_t = typename sub<v1, v2>::type;
01481 
01485         template<typename v1, typename v2>
01486         using mul_t = typename mul<v1, v2>::type;
01487 
01491         template<typename v1, typename v2>
01492         using div_t = typename div<v1, v2>::type;
01493 
01497         template<typename v1, typename v2>
01498         using mod_t = typename remainder<v1, v2>::type;
01499 
01503         template<typename v1, typename v2>
01504         using gt_t = typename gt<v1, v2>::type;
01505 
01509         template<typename v1, typename v2>
01510         static constexpr bool gt_v = gt_t<v1, v2>::value;
01511 
01515         template<typename v1, typename v2>
01516         using lt_t = typename lt<v1, v2>::type;
01517 
01521         template<typename v1, typename v2>
01522         static constexpr bool lt_v = lt_t<v1, v2>::value;
01523 
01527         template<typename v1, typename v2>
01528         using eq_t = typename eq<v1, v2>::type;
01529 
01533         template<typename v1, typename v2>
01534         static constexpr bool eq_v = eq_t<v1, v2>::value;
01535 
01539         template<typename v1, typename v2>
01540         using gcd_t = gcd_t<i32, v1, v2>;
01541 
01544         template<typename v1>
01545         using pos_t = typename pos<v1>::type;
01546 
01549         template<typename v>
01550         static constexpr bool pos_v = pos_t<v>::value;
01551     };
01552 
01555     template<int32_t x>
01556     struct Embed<zpz<x>, i32> {
01559         template <typename val>
01560         using type = i32::val<val::v>;
```

```
01561     };
01562 }  // namespace aerobus
01563
01564 // polynomial
01565 namespace aerobus {
01566     // coeffN x^N + ...
01571     template<typename Ring>
01572     requires IsEuclideanDomain<Ring>
01573     struct polynomial {
01574         static constexpr bool is_field = false;
01575         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01576
01580         template<typename coeffN, typename... coeffs>
01581         struct val {
01583             using ring_type = Ring;
01585             using enclosing_type = polynomial<Ring>;
01587             static constexpr size_t degree = sizeof...(coeffs);
01589             using aN = coeffN;
01591             using strip = val<coeffs...>;
01593             using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01595             static constexpr bool is_zero_v = is_zero_t::value;
01596
01597          private:
01598             template<size_t index, typename E = void>
01599             struct coeff_at {};
01600
01601             template<size_t index>
01602             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))>> {
01603                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01604             };
01605
01606             template<size_t index>
01607             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))>> {
01608                 using type = typename Ring::zero;
01609             };
01610
01611          public:
01614             template<size_t index>
01615             using coeff_at_t = typename coeff_at<index>::type;
01616
01619             static std::string to_string() {
01620                 return string_helper<coeffN, coeffs...>::func();
01621             }
01622
01627             template<typename valueRing>
01628             static constexpr DEVICE INLINED valueRing eval(const valueRing& x) {
01629                 #ifdef WITH_CUDA_FP16
01630                 valueRing start;
01631                 if constexpr (std::is_same_v<valueRing, __half2>) {
01632                     start = __half2(0, 0);
01633                 } else {
01634                     start = static_cast<valueRing>(0);
01635                 }
01636                 #else
01637                 valueRing start = static_cast<valueRing>(0);
01638                 #endif
01639                 return horner_evaluation<valueRing, val>
01640                         ::template inner<0, degree + 1>
01641                         ::func(start, x);
01642             }
01643         };
01644
01647         template<typename coeffN>
01648         struct val<coeffN> {
01650             using ring_type = Ring;
01652             using enclosing_type = polynomial<Ring>;
01654             static constexpr size_t degree = 0;
01655             using aN = coeffN;
01656             using strip = val<coeffN>;
01657             using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01658
01659             static constexpr bool is_zero_v = is_zero_t::value;
01660
01661             template<size_t index, typename E = void>
01662             struct coeff_at {};
01663
01664             template<size_t index>
01665             struct coeff_at<index, std::enable_if_t<(index == 0)>> {
01666                 using type = aN;
01667             };
01668
01669             template<size_t index>
01670             struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)>> {
01671                 using type = typename Ring::zero;
01672             };
01673
01674             template<size_t index>
```

```
01675             using coeff_at_t = typename coeff_at<index>::type;
01676
01677             static std::string to_string() {
01678                 return string_helper<coeffN>::func();
01679             }
01680         };
01681
01683         using zero = val<typename Ring::zero>;
01685         using one = val<typename Ring::one>;
01687         using X = val<typename Ring::one, typename Ring::zero>;
01688
01689     private:
01690         template<typename P, typename E = void>
01691         struct simplify;
01692
01693         template <typename P1, typename P2, typename I>
01694         struct add_low;
01695
01696         template<typename P1, typename P2>
01697         struct add {
01698             using type = typename simplify<typename add_low<
01699             P1,
01700             P2,
01701             internal::make_index_sequence_reverse<
01702             std::max(P1::degree, P2::degree) + 1
01703             »::type>::type;
01704         };
01705
01706         template <typename P1, typename P2, typename I>
01707         struct sub_low;
01708
01709         template <typename P1, typename P2, typename I>
01710         struct mul_low;
01711
01712         template<typename v1, typename v2>
01713         struct mul {
01714                 using type = typename mul_low<
01715                     v1,
01716                     v2,
01717                     internal::make_index_sequence_reverse<
01718                     v1::degree + v2::degree + 1
01719                     »::type;
01720         };
01721
01722         template<typename coeff, size_t deg>
01723         struct monomial;
01724
01725         template<typename v, typename E = void>
01726         struct derive_helper {};
01727
01728         template<typename v>
01729         struct derive_helper<v, std::enable_if_t<v::degree == 0» {
01730             using type = zero;
01731         };
01732
01733         template<typename v>
01734         struct derive_helper<v, std::enable_if_t<v::degree != 0» {
01735             using type = typename add<
01736                 typename derive_helper<typename simplify<typename v::strip>::type>::type,
01737                 typename monomial<
01738                     typename Ring::template mul_t<
01739                         typename v::aN,
01740                         typename Ring::template inject_constant_t<(v::degree)>
01741                     >,
01742                     v::degree - 1
01743                 >::type
01744             >::type;
01745         };
01746
01747         template<typename v1, typename v2, typename E = void>
01748         struct eq_helper {};
01749
01750         template<typename v1, typename v2>
01751         struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
01752             using type = std::false_type;
01753         };
01754
01755
01756         template<typename v1, typename v2>
01757         struct eq_helper<v1, v2, std::enable_if_t<
01758             v1::degree == v2::degree &&
01759             (v1::degree != 0 || v2::degree != 0) &&
01760             std::is_same<
01761             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01762             std::false_type
01763             >::value
01764         >
```

```
01765         > {
01766             using type = std::false_type;
01767         };
01768
01769         template<typename v1, typename v2>
01770         struct eq_helper<v1, v2, std::enable_if_t<
01771             v1::degree == v2::degree &&
01772             (v1::degree != 0 || v2::degree != 0) &&
01773             std::is_same<
01774             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01775             std::true_type
01776             >::value
01777         >> {
01778             using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01779         };
01780
01781         template<typename v1, typename v2>
01782         struct eq_helper<v1, v2, std::enable_if_t<
01783             v1::degree == v2::degree &&
01784             (v1::degree == 0)
01785         >> {
01786             using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01787         };
01788
01789         template<typename v1, typename v2, typename E = void>
01790         struct lt_helper {};
01791
01792         template<typename v1, typename v2>
01793         struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)>> {
01794             using type = std::true_type;
01795         };
01796
01797         template<typename v1, typename v2>
01798         struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)>> {
01799             using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01800         };
01801
01802         template<typename v1, typename v2>
01803         struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)>> {
01804             using type = std::false_type;
01805         };
01806
01807         template<typename v1, typename v2, typename E = void>
01808         struct gt_helper {};
01809
01810         template<typename v1, typename v2>
01811         struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)>> {
01812             using type = std::true_type;
01813         };
01814
01815         template<typename v1, typename v2>
01816         struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)>> {
01817             using type = std::false_type;
01818         };
01819
01820         template<typename v1, typename v2>
01821         struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)>> {
01822             using type = std::false_type;
01823         };
01824
01825         // when high power is zero : strip
01826         template<typename P>
01827         struct simplify<P, std::enable_if_t<
01828             std::is_same<
01829             typename Ring::zero,
01830             typename P::aN
01831             >::value && (P::degree > 0)
01832         >> {
01833             using type = typename simplify<typename P::strip>::type;
01834         };
01835
01836         // otherwise : do nothing
01837         template<typename P>
01838         struct simplify<P, std::enable_if_t<
01839             !std::is_same<
01840             typename Ring::zero,
01841             typename P::aN
01842             >::value && (P::degree > 0)
01843         >> {
01844             using type = P;
01845         };
01846
01847         // do not simplify constants
01848         template<typename P>
01849         struct simplify<P, std::enable_if_t<P::degree == 0>> {
01850             using type = P;
01851         };
```

```
01852
01853         // addition at
01854         template<typename P1, typename P2, size_t index>
01855         struct add_at {
01856             using type =
01857                 typename Ring::template add_t<
01858                     typename P1::template coeff_at_t<index>,
01859                     typename P2::template coeff_at_t<index>>;
01860         };
01861
01862         template<typename P1, typename P2, size_t index>
01863         using add_at_t = typename add_at<P1, P2, index>::type;
01864
01865         template<typename P1, typename P2, std::size_t... I>
01866         struct add_low<P1, P2, std::index_sequence<I...>> {
01867             using type = val<add_at_t<P1, P2, I>...>;
01868         };
01869
01870         // substraction at
01871         template<typename P1, typename P2, size_t index>
01872         struct sub_at {
01873             using type =
01874                 typename Ring::template sub_t<
01875                     typename P1::template coeff_at_t<index>,
01876                     typename P2::template coeff_at_t<index>>;
01877         };
01878
01879         template<typename P1, typename P2, size_t index>
01880         using sub_at_t = typename sub_at<P1, P2, index>::type;
01881
01882         template<typename P1, typename P2, std::size_t... I>
01883         struct sub_low<P1, P2, std::index_sequence<I...>> {
01884             using type = val<sub_at_t<P1, P2, I>...>;
01885         };
01886
01887         template<typename P1, typename P2>
01888         struct sub {
01889             using type = typename simplify<typename sub_low<
01890             P1,
01891             P2,
01892             internal::make_index_sequence_reverse<
01893             std::max(P1::degree, P2::degree) + 1
01894             >::type>::type;
01895         };
01896
01897         // multiplication at
01898         template<typename v1, typename v2, size_t k, size_t index, size_t stop>
01899         struct mul_at_loop_helper {
01900             using type = typename Ring::template add_t<
01901                 typename Ring::template mul_t<
01902                 typename v1::template coeff_at_t<index>,
01903                 typename v2::template coeff_at_t<k - index>
01904                 >,
01905                 typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
01906             >;
01907         };
01908
01909         template<typename v1, typename v2, size_t k, size_t stop>
01910         struct mul_at_loop_helper<v1, v2, k, stop, stop> {
01911             using type = typename Ring::template mul_t<
01912                 typename v1::template coeff_at_t<stop>,
01913                 typename v2::template coeff_at_t<0>>;
01914         };
01915
01916         template <typename v1, typename v2, size_t k, typename E = void>
01917         struct mul_at {};
01918
01919         template<typename v1, typename v2, size_t k>
01920         struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)>> {
01921             using type = typename Ring::zero;
01922         };
01923
01924         template<typename v1, typename v2, size_t k>
01925         struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)>> {
01926             using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01927         };
01928
01929         template<typename P1, typename P2, size_t index>
01930         using mul_at_t = typename mul_at<P1, P2, index>::type;
01931
01932         template<typename P1, typename P2, std::size_t... I>
01933         struct mul_low<P1, P2, std::index_sequence<I...>> {
01934             using type = val<mul_at_t<P1, P2, I>...>;
01935         };
01936
01937         // division helper
01938         template< typename A, typename B, typename Q, typename R, typename E = void>
```

```
01939         struct div_helper {};
01940
01941         template<typename A, typename B, typename Q, typename R>
01942         struct div_helper<A, B, Q, R, std::enable_if_t<
01943             (R::degree < B::degree) ||
01944             (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
01945             using q_type = Q;
01946             using mod_type = R;
01947             using gcd_type = B;
01948         };
01949
01950         template<typename A, typename B, typename Q, typename R>
01951         struct div_helper<A, B, Q, R, std::enable_if_t<
01952             (R::degree >= B::degree) &&
01953             !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
01954          private: // NOLINT
01955             using rN = typename R::aN;
01956             using bN = typename B::aN;
01957             using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
      B::degree>::type;
01958             using rr = typename sub<R, typename mul<pT, B>::type>::type;
01959             using qq = typename add<Q, pT>::type;
01960
01961          public:
01962             using q_type = typename div_helper<A, B, qq, rr>::q_type;
01963             using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
01964             using gcd_type = rr;
01965         };
01966
01967         template<typename A, typename B>
01968         struct div {
01969             static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
01970             using q_type = typename div_helper<A, B, zero, A>::q_type;
01971             using m_type = typename div_helper<A, B, zero, A>::mod_type;
01972         };
01973
01974         template<typename P>
01975         struct make_unit {
01976             using type = typename div<P, val<typename P::aN>>::q_type;
01977         };
01978
01979         template<typename coeff, size_t deg>
01980         struct monomial {
01981             using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
01982         };
01983
01984         template<typename coeff>
01985         struct monomial<coeff, 0> {
01986             using type = val<coeff>;
01987         };
01988
01989         template<typename valueRing, typename P>
01990         struct horner_evaluation {
01991             template<size_t index, size_t stop>
01992             struct inner {
01993                 static constexpr DEVICE INLINED valueRing func(const valueRing& accum, const
      valueRing& x) {
01994                     return horner_evaluation<valueRing, P>::template inner<index + 1, stop>::func(
01995                         internal::fma_helper<valueRing>::eval(
01996                             x,
01997                             accum,
01998                             P::template coeff_at_t<P::degree - index>::template get<valueRing>()), x);
01999                 }
02000             };
02001
02002             template<size_t stop>
02003             struct inner<stop, stop> {
02004                 static constexpr DEVICE INLINED valueRing func(const valueRing& accum, const
      valueRing& x) {
02005                     return accum;
02006                 }
02007             };
02008         };
02009
02010         template<typename coeff, typename... coeffs>
02011         struct string_helper {
02012             static std::string func() {
02013                 std::string tail = string_helper<coeffs...>::func();
02014                 std::string result = "";
02015                 if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
02016                     return tail;
02017                 } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
02018                     if (sizeof...(coeffs) == 1) {
02019                         result += "x";
02020                     } else {
02021                         result += "x^" + std::to_string(sizeof...(coeffs));
02022                     }
```

```
02023                 } else {
02024                     if (sizeof...(coeffs) == 1) {
02025                         result += coeff::to_string() + " x";
02026                     } else {
02027                         result += coeff::to_string()
02028                                 + " x^" + std::to_string(sizeof...(coeffs));
02029                     }
02030                 }
02031 
02032                 if (!tail.empty()) {
02033                     result += " + " + tail;
02034                 }
02035 
02036                 return result;
02037             }
02038         };
02039 
02040         template<typename coeff>
02041         struct string_helper<coeff> {
02042             static std::string func() {
02043                 if (!std::is_same<coeff, typename Ring::zero>::value) {
02044                     return coeff::to_string();
02045                 } else {
02046                     return "";
02047                 }
02048             }
02049         };
02050 
02051     public:
02054         template<typename P>
02055         using simplify_t = typename simplify<P>::type;
02056 
02060         template<typename v1, typename v2>
02061         using add_t = typename add<v1, v2>::type;
02062 
02066         template<typename v1, typename v2>
02067         using sub_t = typename sub<v1, v2>::type;
02068 
02072         template<typename v1, typename v2>
02073         using mul_t = typename mul<v1, v2>::type;
02074 
02078         template<typename v1, typename v2>
02079         using eq_t = typename eq_helper<v1, v2>::type;
02080 
02084         template<typename v1, typename v2>
02085         using lt_t = typename lt_helper<v1, v2>::type;
02086 
02090         template<typename v1, typename v2>
02091         using gt_t = typename gt_helper<v1, v2>::type;
02092 
02096         template<typename v1, typename v2>
02097         using div_t = typename div<v1, v2>::q_type;
02098 
02102         template<typename v1, typename v2>
02103         using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
02104 
02108         template<typename coeff, size_t deg>
02109         using monomial_t = typename monomial<coeff, deg>::type;
02110 
02113         template<typename v>
02114         using derive_t = typename derive_helper<v>::type;
02115 
02118         template<typename v>
02119         using pos_t = typename Ring::template pos_t<typename v::aN>;
02120 
02123         template<typename v>
02124         static constexpr bool pos_v = pos_t<v>::value;
02125 
02129         template<typename v1, typename v2>
02130         using gcd_t = std::conditional_t<
02131             Ring::is_euclidean_domain,
02132             typename make_unit<gcd_t<polynomial<Ring>, v1, v2>>::type,
02133             void>;
02134 
02138         template<auto x>
02139         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
02140 
02144         template<typename v>
02145         using inject_ring_t = val<v>;
02146     };
02147 }  // namespace aerobus
02148 
02149 // fraction field
02150 namespace aerobus {
02151     namespace internal {
02152         template<typename Ring, typename E = void>
02153         requires IsEuclideanDomain<Ring>
```

```
02154         struct _FractionField {};
02155
02156         template<typename Ring>
02157         requires IsEuclideanDomain<Ring>
02158         struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain» {
02160             static constexpr bool is_field = true;
02161             static constexpr bool is_euclidean_domain = true;
02162
02163          private:
02164             template<typename val1, typename val2, typename E = void>
02165             struct to_string_helper {};
02166
02167             template<typename val1, typename val2>
02168             struct to_string_helper <val1, val2,
02169                 std::enable_if_t<
02170                 Ring::template eq_t<
02171                 val2, typename Ring::one
02172                 >::value
02173                 >
02174             > {
02175                 static std::string func() {
02176                     return val1::to_string();
02177                 }
02178             };
02179
02180             template<typename val1, typename val2>
02181             struct to_string_helper<val1, val2,
02182                 std::enable_if_t<
02183                 !Ring::template eq_t<
02184                 val2,
02185                 typename Ring::one
02186                 >::value
02187                 >
02188             > {
02189                 static std::string func() {
02190                     return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
02191                 }
02192             };
02193
02194          public:
02198             template<typename val1, typename val2>
02199             struct val {
02201                 using x = val1;
02203                 using y = val2;
02205                 using is_zero_t = typename val1::is_zero_t;
02207                 static constexpr bool is_zero_v = val1::is_zero_t::value;
02208
02210                 using ring_type = Ring;
02211                 using enclosing_type = _FractionField<Ring>;
02212
02215                  static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
02216
02220                 template<typename valueType>
02221                 static constexpr INLINED DEVICE valueType get() {
02222                     return internal::staticcast<valueType, typename ring_type::inner_type>::template
      func<x::v>() /
02223                         internal::staticcast<valueType, typename ring_type::inner_type>::template
      func<y::v>();
02224                 }
02225
02228                 static std::string to_string() {
02229                     return to_string_helper<val1, val2>::func();
02230                 }
02231
02236                 template<typename valueRing>
02237                 static constexpr DEVICE INLINED valueRing eval(const valueRing& v) {
02238                     return x::eval(v) / y::eval(v);
02239                 }
02240             };
02241
02243             using zero = val<typename Ring::zero, typename Ring::one>;
02245             using one = val<typename Ring::one, typename Ring::one>;
02246
02249             template<typename v>
02250             using inject_t = val<v, typename Ring::one>;
02251
02254             template<auto x>
02255             using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
      Ring::one>;
02256
02259             template<typename v>
02260             using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
02261
02263             using ring_type = Ring;
02264
02265          private:
02266             template<typename v, typename E = void>
```

```
02267             struct simplify {};
02268
02269             // x = 0
02270             template<typename v>
02271             struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {
02272                 using type = typename _FractionField<Ring>::zero;
02273             };
02274
02275             // x != 0
02276             template<typename v>
02277             struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value» {
02278              private:
02279                 using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
02280                 using newx = typename Ring::template div_t<typename v::x, _gcd>;
02281                 using newy = typename Ring::template div_t<typename v::y, _gcd>;
02282
02283                 using posx = std::conditional_t<
02284                                 !Ring::template pos_v<newy>,
02285                                 typename Ring::template sub_t<typename Ring::zero, newx>,
02286                                 newx>;
02287                 using posy = std::conditional_t<
02288                                 !Ring::template pos_v<newy>,
02289                                 typename Ring::template sub_t<typename Ring::zero, newy>,
02290                                 newy>;
02291              public:
02292                 using type = typename _FractionField<Ring>::template val<posx, posy>;
02293             };
02294
02295          public:
02298             template<typename v>
02299             using simplify_t = typename simplify<v>::type;
02300
02301          private:
02302             template<typename v1, typename v2>
02303             struct add {
02304              private:
02305                 using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02306                 using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02307                 using dividend = typename Ring::template add_t<a, b>;
02308                 using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02309                 using g = typename Ring::template gcd_t<dividend, diviser>;
02310
02311              public:
02312                 using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
      diviser»;
02313             };
02314
02315             template<typename v>
02316             struct pos {
02317                 using type = std::conditional_t<
02318                     (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
02319                     (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
02320                     std::true_type,
02321                     std::false_type>;
02322             };
02323
02324             template<typename v1, typename v2>
02325             struct sub {
02326              private:
02327                 using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02328                 using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02329                 using dividend = typename Ring::template sub_t<a, b>;
02330                 using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02331                 using g = typename Ring::template gcd_t<dividend, diviser>;
02332
02333              public:
02334                 using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
      diviser»;
02335             };
02336
02337             template<typename v1, typename v2>
02338             struct mul {
02339              private:
02340                 using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
02341                 using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02342
02343              public:
02344                 using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
02345             };
02346
02347             template<typename v1, typename v2, typename E = void>
02348             struct div {};
02349
02350             template<typename v1, typename v2>
02351             struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
      _FractionField<Ring>::zero>::value»  {
02352              private:
```

```
02353                 using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02354                 using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02355
02356              public:
02357                 using type = typename _FractionField<Ring>::template simplify_t<val<a, b>>;
02358             };
02359
02360             template<typename v1, typename v2>
02361             struct div<v1, v2, std::enable_if_t<
02362                 std::is_same<zero, v1>::value && std::is_same<v2, zero>::value>  {
02363                 using type = one;
02364             };
02365
02366             template<typename v1, typename v2>
02367             struct eq {
02368                 using type = std::conditional_t<
02369                         std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
02370                         std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
02371                     std::true_type,
02372                     std::false_type>;
02373             };
02374
02375             template<typename v1, typename v2, typename E = void>
02376             struct gt;
02377
02378             template<typename v1, typename v2>
02379             struct gt<v1, v2, std::enable_if_t<
02380                 (eq<v1, v2>::type::value)
02381                 >> {
02382                 using type = std::false_type;
02383             };
02384
02385             template<typename v1, typename v2>
02386             struct gt<v1, v2, std::enable_if_t<
02387                 (!eq<v1, v2>::type::value) &&
02388                 (!pos<v1>::type::value) && (!pos<v2>::type::value)
02389                 >> {
02390                 using type = typename gt<
02391                     typename sub<zero, v1>::type, typename sub<zero, v2>::type
02392                 >::type;
02393             };
02394
02395             template<typename v1, typename v2>
02396             struct gt<v1, v2, std::enable_if_t<
02397                 (!eq<v1, v2>::type::value) &&
02398                 (pos<v1>::type::value) && (!pos<v2>::type::value)
02399                 >> {
02400                 using type = std::true_type;
02401             };
02402
02403             template<typename v1, typename v2>
02404             struct gt<v1, v2, std::enable_if_t<
02405                 (!eq<v1, v2>::type::value) &&
02406                 (!pos<v1>::type::value) && (pos<v2>::type::value)
02407                 >> {
02408                 using type = std::false_type;
02409             };
02410
02411             template<typename v1, typename v2>
02412             struct gt<v1, v2, std::enable_if_t<
02413                 (!eq<v1, v2>::type::value) &&
02414                 (pos<v1>::type::value) && (pos<v2>::type::value)
02415                 >> {
02416                 using type = typename Ring::template gt_t<
02417                     typename Ring::template mul_t<v1::x, v2::y>,
02418                     typename Ring::template mul_t<v2::y, v2::x>
02419                 >;
02420             };
02421
02422          public:
02427             template<typename v1, typename v2>
02428             using add_t = typename add<v1, v2>::type;
02429
02434             template<typename v1, typename v2>
02435             using mod_t = zero;
02436
02441             template<typename v1, typename v2>
02442             using gcd_t = v1;
02443
02447             template<typename v1, typename v2>
02448             using sub_t = typename sub<v1, v2>::type;
02449
02453             template<typename v1, typename v2>
02454             using mul_t = typename mul<v1, v2>::type;
02455
02459             template<typename v1, typename v2>
02460             using div_t = typename div<v1, v2>::type;
```

```
02461
02465             template<typename v1, typename v2>
02466             using eq_t = typename eq<v1, v2>::type;
02467
02471             template<typename v1, typename v2>
02472             static constexpr bool eq_v = eq<v1, v2>::type::value;
02473
02477             template<typename v1, typename v2>
02478             using gt_t = typename gt<v1, v2>::type;
02479
02483             template<typename v1, typename v2>
02484             static constexpr bool gt_v = gt<v1, v2>::type::value;
02485
02488             template<typename v1>
02489             using pos_t = typename pos<v1>::type;
02490
02493             template<typename v>
02494             static constexpr bool pos_v = pos_t<v>::value;
02495         };
02496
02497         template<typename Ring, typename E = void>
02498         requires IsEuclideanDomain<Ring>
02499         struct FractionFieldImpl {};
02500
02501         // fraction field of a field is the field itself
02502         template<typename Field>
02503         requires IsEuclideanDomain<Field>
02504         struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {
02505             using type = Field;
02506             template<typename v>
02507             using inject_t = v;
02508         };
02509
02510         // fraction field of a ring is the actual fraction field
02511         template<typename Ring>
02512         requires IsEuclideanDomain<Ring>
02513         struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {
02514             using type = _FractionField<Ring>;
02515         };
02516     }  // namespace internal
02517
02521     template<typename Ring>
02522     requires IsEuclideanDomain<Ring>
02523     using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02524
02527     template<typename Ring>
02528     struct Embed<Ring, FractionField<Ring» {
02531         template<typename v>
02532         using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02533     };
02534 }  // namespace aerobus
02535
02536
02537 // short names for common types
02538 namespace aerobus {
02542     template<typename X, typename Y>
02543     requires IsRing<typename X::enclosing_type> &&
02544         (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02545     using add_t = typename X::enclosing_type::template add_t<X, Y>;
02546
02550     template<typename X, typename Y>
02551     requires IsRing<typename X::enclosing_type> &&
02552         (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02553     using sub_t = typename X::enclosing_type::template sub_t<X, Y>;
02554
02558     template<typename X, typename Y>
02559     requires IsRing<typename X::enclosing_type> &&
02560         (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02561     using mul_t = typename X::enclosing_type::template mul_t<X, Y>;
02562
02566     template<typename X, typename Y>
02567     requires IsEuclideanDomain<typename X::enclosing_type> &&
02568         (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02569     using div_t = typename X::enclosing_type::template div_t<X, Y>;
02570
02573     using q32 = FractionField<i32>;
02574
02577     using fpq32 = FractionField<polynomial<q32>>;
02578
02581     using q64 = FractionField<i64>;
02582
02584     using pi64 = polynomial<i64>;
02585
02587     using pq64 = polynomial<q64>;
02588
02590     using fpq64 = FractionField<polynomial<q64>>;
02591
```

```
02596     template<typename Ring, typename v1, typename v2>
02597     using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02598
02605     template<typename v>
02606     using embed_int_poly_in_fractions_t =
02607             typename Embed<
02608                 polynomial<typename v::ring_type>,
02609                 polynomial<FractionField<typename v::ring_type>»::template type<v>;
02610
02614     template<int64_t p, int64_t q>
02615     using make_q64_t = typename q64::template simplify_t<
02616                 typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02617
02621     template<int32_t p, int32_t q>
02622     using make_q32_t = typename q32::template simplify_t<
02623                 typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>»;
02624
02629     template<typename Ring, typename v1, typename v2>
02630     using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02635     template<typename Ring, typename v1, typename v2>
02636     using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02637
02639     template<>
02640     struct Embed<q32, q64> {
02643         template<typename v>
02644         using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02645     };
02646
02650     template<typename Small, typename Large>
02651     struct Embed<polynomial<Small>, polynomial<Large» {
02652      private:
02653         template<typename v, typename i>
02654         struct at_low;
02655
02656         template<typename v, size_t i>
02657         struct at_index {
02658             using type = typename Embed<Small, Large>::template
      type<typename v::template coeff_at_t<i>>;
02659         };
02660
02661         template<typename v, size_t... Is>
02662         struct at_low<v, std::index_sequence<Is...» {
02663             using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
02664         };
02665
02666      public:
02669         template<typename v>
02670         using type = typename
      at_low<v, typename internal::make_index_sequence_reverse<v::degree + 1>>::type;
02671     };
02672
02676     template<typename Ring, auto... xs>
02677     using make_int_polynomial_t = typename polynomial<Ring>::template val<
02678             typename Ring::template inject_constant_t<xs>...>;
02679
02683     template<typename Ring, auto... xs>
02684     using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<
02685             typename FractionField<Ring>::template inject_constant_t<xs>...>;
02686 }  // namespace aerobus
02687
02688 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02689 namespace aerobus {
02690     namespace internal {
02691         template<typename T, size_t x, typename E = void>
02692         struct factorial {};
02693
02694         template<typename T, size_t x>
02695         struct factorial<T, x, std::enable_if_t<(x > 0)» {
02696         private:
02697             template<typename, size_t, typename>
02698             friend struct factorial;
02699         public:
02700             using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
      x - 1>::type>;
02701             static constexpr typename T::inner_type value = type::template
      get<typename T::inner_type>();
02702         };
02703
02704         template<typename T>
02705         struct factorial<T, 0> {
02706          public:
02707             using type = typename T::one;
02708             static constexpr typename T::inner_type value = type::template
      get<typename T::inner_type>();
02709         };
02710     }  // namespace internal
02711
```

```
02715     template<typename T, size_t i>
02716     using factorial_t = typename internal::factorial<T, i>::type;
02717
02721     template<typename T, size_t i>
02722     inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02723
02724     namespace internal {
02725         template<typename T, size_t k, size_t n, typename E = void>
02726         struct combination_helper {};
02727
02728         template<typename T, size_t k, size_t n>
02729         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
02730             using type = typename FractionField<T>::template mul_t<
02731                 typename combination_helper<T, k - 1, n - 1>::type,
02732                 makefraction_t<T, typename T::template val<n>, typename T::template val<k>»;
02733         };
02734
02735         template<typename T, size_t k, size_t n>
02736         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)» {
02737             using type = typename combination_helper<T, n - k, n>::type;
02738         };
02739
02740         template<typename T, size_t n>
02741         struct combination_helper<T, 0, n> {
02742             using type = typename FractionField<T>::one;
02743         };
02744
02745         template<typename T, size_t k, size_t n>
02746         struct combination {
02747             using type = typename internal::combination_helper<T, k, n>::type::x;
02748             static constexpr typename T::inner_type value =
02749                         internal::combination_helper<T, k, n>::type::template
      get<typename T::inner_type>();
02750         };
02751     }  // namespace internal
02752
02755     template<typename T, size_t k, size_t n>
02756     using combination_t = typename internal::combination<T, k, n>::type;
02757
02762     template<typename T, size_t k, size_t n>
02763     inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02764
02765     namespace internal {
02766         template<typename T, size_t m>
02767         struct bernoulli;
02768
02769         template<typename T, typename accum, size_t k, size_t m>
02770         struct bernoulli_helper {
02771             using type = typename bernoulli_helper<
02772                 T,
02773                 addfractions_t<T,
02774                     accum,
02775                     mulfractions_t<T,
02776                         makefraction_t<T,
02777                             combination_t<T, k, m + 1>,
02778                             typename T::one>,
02779                         typename bernoulli<T, k>::type
02780                     >
02781                 >,
02782                 k + 1,
02783                 m>::type;
02784         };
02785
02786         template<typename T, typename accum, size_t m>
02787         struct bernoulli_helper<T, accum, m, m> {
02788             using type = accum;
02789         };
02790
02791
02792
02793         template<typename T, size_t m>
02794         struct bernoulli {
02795             using type = typename FractionField<T>::template mul_t<
02796                 typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02797                 makefraction_t<T,
02798                 typename T::template val<static_cast<typename T::inner_type>(-1)>,
02799                 typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02800                 >
02801             >;
02802
02803             template<typename floatType>
02804             static constexpr floatType value = type::template get<floatType>();
02805         };
02806
02807         template<typename T>
02808         struct bernoulli<T, 0> {
02809             using type = typename FractionField<T>::one;
```

```
02810
02811             template<typename floatType>
02812             static constexpr floatType value = type::template get<floatType>();
02813         };
02814     }  // namespace internal
02815
02819     template<typename T, size_t n>
02820     using bernoulli_t = typename internal::bernoulli<T, n>::type;
02821
02826     template<typename FloatType, typename T, size_t n >
02827     inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02828
02829     // bell numbers
02830     namespace internal {
02831         template<typename T, size_t n, typename E = void>
02832         struct bell_helper;
02833
02834         template <typename T, size_t n>
02835         struct bell_helper<T, n, std::enable_if_t<(n > 1)» {
02836             template<typename accum, size_t i, size_t stop>
02837             struct sum_helper {
02838              private:
02839                 using left = typename T::template mul_t<
02840                             combination_t<T, i, n-1>,
02841                             typename bell_helper<T, i>::type>;
02842                 using new_accum = typename T::template add_t<accum, left>;
02843              public:
02844                 using type = typename sum_helper<new_accum, i+1, stop>::type;
02845             };
02846
02847             template<typename accum, size_t stop>
02848             struct sum_helper<accum, stop, stop> {
02849                 using type = accum;
02850             };
02851
02852             using type = typename sum_helper<typename T::zero, 0, n>::type;
02853         };
02854
02855         template<typename T>
02856         struct bell_helper<T, 0> {
02857             using type = typename T::one;
02858         };
02859
02860         template<typename T>
02861         struct bell_helper<T, 1> {
02862             using type = typename T::one;
02863         };
02864     }  // namespace internal
02865
02869     template<typename T, size_t n>
02870     using bell_t = typename internal::bell_helper<T, n>::type;
02871
02875     template<typename T, size_t n>
02876     static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
02877
02878     namespace internal {
02879         template<typename T, int k, typename E = void>
02880         struct alternate {};
02881
02882         template<typename T, int k>
02883         struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
02884             using type = typename T::one;
02885             static constexpr typename T::inner_type value = type::template
      get<typename T::inner_type>();
02886         };
02887
02888         template<typename T, int k>
02889         struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
02890             using type = typename T::template sub_t<typename T::zero, typename T::one>;
02891             static constexpr typename T::inner_type value = type::template
      get<typename T::inner_type>();
02892         };
02893     }  // namespace internal
02894
02897     template<typename T, int k>
02898     using alternate_t = typename internal::alternate<T, k>::type;
02899
02900     namespace internal {
02901         template<typename T, int n, int k, typename E = void>
02902         struct stirling_helper {};
02903
02904         template<typename T>
02905         struct stirling_helper<T, 0, 0> {
02906             using type = typename T::one;
02907         };
02908
02909         template<typename T, int n>
```

```
02910         struct stirling_helper<T, n, 0, std::enable_if_t<(n > 0)» {
02911             using type = typename T::zero;
02912         };
02913 
02914         template<typename T, int n>
02915         struct stirling_helper<T, 0, n, std::enable_if_t<(n > 0)» {
02916             using type = typename T::zero;
02917         };
02918 
02919         template<typename T, int n, int k>
02920         struct stirling_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)» {
02921             using type = typename T::template sub_t<
02922                             typename stirling_helper<T, n-1, k-1>::type,
02923                             typename T::template mul_t<
02924                                 typename T::template inject_constant_t<n-1>,
02925                                 typename stirling_helper<T, n-1, k>::type
02926                             »;
02927         };
02928     }  // namespace internal
02929 
02934     template<typename T, int n, int k>
02935     using stirling_signed_t = typename internal::stirling_helper<T, n, k>::type;
02936 
02941     template<typename T, int n, int k>
02942     using stirling_unsigned_t = abs_t<typename internal::stirling_helper<T, n, k>::type>;
02943 
02948     template<typename T, int n, int k>
02949     static constexpr typename T::inner_type stirling_signed_v = stirling_signed_t<T, n, k>::v;
02950 
02951 
02956     template<typename T, int n, int k>
02957     static constexpr typename T::inner_type stirling_unsigned_v = stirling_unsigned_t<T, n, k>::v;
02958 
02961     template<typename T, size_t k>
02962     inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
02963 
02964     namespace internal {
02965         template<typename T>
02966         struct pow_scalar {
02967             template<size_t p>
02968             static constexpr DEVICE INLINED T func(const T& x) { return p == 0 ? static_cast<T>(1) :
02969                 p % 2 == 0 ? func<p/2>(x) * func<p/2>(x) :
02970                 x * func<p/2>(x) * func<p/2>(x);
02971             }
02972         };
02973 
02974         template<typename T, typename p, size_t n, typename E = void>
02975         requires IsEuclideanDomain<T>
02976         struct pow;
02977 
02978         template<typename T, typename p, size_t n>
02979         struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)» {
02980             using type = typename T::template mul_t<
02981                 typename pow<T, p, n/2>::type,
02982                 typename pow<T, p, n/2>::type
02983             >;
02984         };
02985 
02986         template<typename T, typename p, size_t n>
02987         struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)» {
02988             using type = typename T::template mul_t<
02989                 p,
02990                 typename T::template mul_t<
02991                     typename pow<T, p, n/2>::type,
02992                     typename pow<T, p, n/2>::type
02993                 >
02994             >;
02995         };
02996 
02997         template<typename T, typename p, size_t n>
02998         struct pow<T, p, n, std::enable_if_t<n == 0» { using type = typename T::one; };
02999     }  // namespace internal
03000 
03005     template<typename T, typename p, size_t n>
03006     using pow_t = typename internal::pow<T, p, n>::type;
03007 
03012     template<typename T, typename p, size_t n>
03013     static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
03014 
03015     template<typename T, size_t p>
03016     static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
    internal::pow_scalar<T>::template func<p>(x); }
03017 
03018     namespace internal {
03019         template<typename, template<typename, size_t> typename, class>
03020         struct make_taylor_impl;
03021 
```

```
03022         template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
03023         struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...>> {
03024             using type = typename polynomial<FractionField<T>>::template
      val<typename coeff_at<T, Is>::type...>;
03025         };
03026     }
03027
03032     template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
03033     using taylor = typename internal::make_taylor_impl<
03034         T,
03035         coeff_at,
03036         internal::make_index_sequence_reverse<deg + 1>>::type;
03037
03038     namespace internal {
03039         template<typename T, size_t i>
03040         struct exp_coeff {
03041             using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03042         };
03043
03044         template<typename T, size_t i, typename E = void>
03045         struct sin_coeff_helper {};
03046
03047         template<typename T, size_t i>
03048         struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
03049             using type = typename FractionField<T>::zero;
03050         };
03051
03052         template<typename T, size_t i>
03053         struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
03054             using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03055         };
03056
03057         template<typename T, size_t i>
03058         struct sin_coeff {
03059             using type = typename sin_coeff_helper<T, i>::type;
03060         };
03061
03062         template<typename T, size_t i, typename E = void>
03063         struct sh_coeff_helper {};
03064
03065         template<typename T, size_t i>
03066         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
03067             using type = typename FractionField<T>::zero;
03068         };
03069
03070         template<typename T, size_t i>
03071         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
03072             using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03073         };
03074
03075         template<typename T, size_t i>
03076         struct sh_coeff {
03077             using type = typename sh_coeff_helper<T, i>::type;
03078         };
03079
03080         template<typename T, size_t i, typename E = void>
03081         struct cos_coeff_helper {};
03082
03083         template<typename T, size_t i>
03084         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
03085             using type = typename FractionField<T>::zero;
03086         };
03087
03088         template<typename T, size_t i>
03089         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
03090             using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03091         };
03092
03093         template<typename T, size_t i>
03094         struct cos_coeff {
03095             using type = typename cos_coeff_helper<T, i>::type;
03096         };
03097
03098         template<typename T, size_t i, typename E = void>
03099         struct cosh_coeff_helper {};
03100
03101         template<typename T, size_t i>
03102         struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
03103             using type = typename FractionField<T>::zero;
03104         };
03105
03106         template<typename T, size_t i>
03107         struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
03108             using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03109         };
03110
03111         template<typename T, size_t i>
```

```
03112         struct cosh_coeff {
03113             using type = typename cosh_coeff_helper<T, i>::type;
03114         };
03115
03116         template<typename T, size_t i>
03117         struct geom_coeff { using type = typename FractionField<T>::one; };
03118
03119
03120         template<typename T, size_t i, typename E = void>
03121         struct atan_coeff_helper;
03122
03123         template<typename T, size_t i>
03124         struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03125             using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i>>;
03126         };
03127
03128         template<typename T, size_t i>
03129         struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03130             using type = typename FractionField<T>::zero;
03131         };
03132
03133         template<typename T, size_t i>
03134         struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
03135
03136         template<typename T, size_t i, typename E = void>
03137         struct asin_coeff_helper;
03138
03139         template<typename T, size_t i>
03140         struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03141             using type = makefraction_t<T,
03142                 factorial_t<T, i - 1>,
03143                 typename T::template mul_t<
03144                     typename T::template val<i>,
03145                     T::template mul_t<
03146                         pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
03147                         pow<T, factorial_t<T, i / 2>, 2
03148                     >
03149                 >
03150                 »;
03151         };
03152
03153         template<typename T, size_t i>
03154         struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03155             using type = typename FractionField<T>::zero;
03156         };
03157
03158         template<typename T, size_t i>
03159         struct asin_coeff {
03160             using type = typename asin_coeff_helper<T, i>::type;
03161         };
03162
03163         template<typename T, size_t i>
03164         struct lnp1_coeff {
03165             using type = makefraction_t<T,
03166                 alternate_t<T, i + 1>,
03167                 typename T::template val<i>>;
03168         };
03169
03170         template<typename T>
03171         struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
03172
03173         template<typename T, size_t i, typename E = void>
03174         struct asinh_coeff_helper;
03175
03176         template<typename T, size_t i>
03177         struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03178             using type = makefraction_t<T,
03179                 typename T::template mul_t<
03180                     alternate_t<T, i / 2>,
03181                     factorial_t<T, i - 1>
03182                 >,
03183                 typename T::template mul_t<
03184                     typename T::template mul_t<
03185                         typename T::template val<i>,
03186                         pow_t<T, factorial_t<T, i / 2>, 2>
03187                     >,
03188                     pow_t<T, typename T::template inject_constant_t<4>, i / 2>
03189                 >
03190             >;
03191         };
03192
03193         template<typename T, size_t i>
03194         struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03195             using type = typename FractionField<T>::zero;
03196         };
03197
03198         template<typename T, size_t i>
```

```
03199         struct asinh_coeff {
03200             using type = typename asinh_coeff_helper<T, i>::type;
03201         };
03202 
03203         template<typename T, size_t i, typename E = void>
03204         struct atanh_coeff_helper;
03205 
03206         template<typename T, size_t i>
03207         struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
03208             // 1/i
03209             using type = typename FractionField<T>:: template val<
03210                 typename T::one,
03211                 typename T::template inject_constant_t<i>>;
03212         };
03213 
03214         template<typename T, size_t i>
03215         struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
03216             using type = typename FractionField<T>::zero;
03217         };
03218 
03219         template<typename T, size_t i>
03220         struct atanh_coeff {
03221             using type = typename atanh_coeff_helper<T, i>::type;
03222         };
03223 
03224         template<typename T, size_t i, typename E = void>
03225         struct tan_coeff_helper;
03226 
03227         template<typename T, size_t i>
03228         struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0>> {
03229             using type = typename FractionField<T>::zero;
03230         };
03231 
03232         template<typename T, size_t i>
03233         struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0>> {
03234         private:
03235             // 4^((i+1)/2)
03236             using _4p = typename FractionField<T>::template inject_t<
03237                 pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2>>;
03238             // 4^((i+1)/2) - 1
03239             using _4pm1 = typename FractionField<T>::template
      sub_t<_4p, typename FractionField<T>::one>;
03240             // (-1)^((i-1)/2)
03241             using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2>>;
03242             using dividend = typename FractionField<T>::template mul_t<
03243                 altp,
03244                 FractionField<T>::template mul_t<
03245                 _4p,
03246                 FractionField<T>::template mul_t<
03247                 _4pm1,
03248                 bernoulli_t<T, (i + 1)>
03249                 >
03250                 >
03251             >;
03252         public:
03253             using type = typename FractionField<T>::template div_t<dividend,
03254                 typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>>;
03255         };
03256 
03257         template<typename T, size_t i>
03258         struct tan_coeff {
03259             using type = typename tan_coeff_helper<T, i>::type;
03260         };
03261 
03262         template<typename T, size_t i, typename E = void>
03263         struct tanh_coeff_helper;
03264 
03265         template<typename T, size_t i>
03266         struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0>> {
03267             using type = typename FractionField<T>::zero;
03268         };
03269 
03270         template<typename T, size_t i>
03271         struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0>> {
03272         private:
03273             using _4p = typename FractionField<T>::template inject_t<
03274                 pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2>>;
03275             using _4pm1 = typename FractionField<T>::template
      sub_t<_4p, typename FractionField<T>::one>;
03276             using dividend =
03277                 typename FractionField<T>::template mul_t<
03278                     _4p,
03279                     typename FractionField<T>::template mul_t<
03280                         _4pm1,
03281                         bernoulli_t<T, (i + 1)>>>::type;
03282         public:
03283             using type = typename FractionField<T>::template div_t<dividend,
```

```
03284                 FractionField<T>::template inject_t<factorial_t<T, i + 1>»;
03285         };
03286
03287         template<typename T, size_t i>
03288         struct tanh_coeff {
03289             using type = typename tanh_coeff_helper<T, i>::type;
03290         };
03291     }  // namespace internal
03292
03296     template<typename Integers, size_t deg>
03297     using exp = taylor<Integers, internal::exp_coeff, deg>;
03298
03302     template<typename Integers, size_t deg>
03303     using expm1 = typename polynomial<FractionField<Integers>>::template sub_t<
03304         exp<Integers, deg>,
03305         typename polynomial<FractionField<Integers>>::one>;
03306
03310     template<typename Integers, size_t deg>
03311     using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
03312
03316     template<typename Integers, size_t deg>
03317     using atan = taylor<Integers, internal::atan_coeff, deg>;
03318
03322     template<typename Integers, size_t deg>
03323     using sin = taylor<Integers, internal::sin_coeff, deg>;
03324
03328     template<typename Integers, size_t deg>
03329     using sinh = taylor<Integers, internal::sh_coeff, deg>;
03330
03335     template<typename Integers, size_t deg>
03336     using cosh = taylor<Integers, internal::cosh_coeff, deg>;
03337
03342     template<typename Integers, size_t deg>
03343     using cos = taylor<Integers, internal::cos_coeff, deg>;
03344
03349     template<typename Integers, size_t deg>
03350     using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
03351
03356     template<typename Integers, size_t deg>
03357     using asin = taylor<Integers, internal::asin_coeff, deg>;
03358
03363     template<typename Integers, size_t deg>
03364     using asinh = taylor<Integers, internal::asinh_coeff, deg>;
03365
03370     template<typename Integers, size_t deg>
03371     using atanh = taylor<Integers, internal::atanh_coeff, deg>;
03372
03377     template<typename Integers, size_t deg>
03378     using tan = taylor<Integers, internal::tan_coeff, deg>;
03379
03384     template<typename Integers, size_t deg>
03385     using tanh = taylor<Integers, internal::tanh_coeff, deg>;
03386 }  // namespace aerobus
03387
03388 // continued fractions
03389 namespace aerobus {
03392     template<int64_t... values>
03393     struct ContinuedFraction {};
03394
03397     template<int64_t a0>
03398     struct ContinuedFraction<a0> {
03400         using type = typename q64::template inject_constant_t<a0>;
03402         static constexpr double val = static_cast<double>(a0);
03403     };
03404
03408     template<int64_t a0, int64_t... rest>
03409     struct ContinuedFraction<a0, rest...> {
03411         using type = q64::template add_t<
03412                 typename q64::template inject_constant_t<a0>,
03413                 typename q64::template div_t<
03414                     typename q64::one,
03415                     typename ContinuedFraction<rest...>::type
03416                 »;
03417
03419         static constexpr double val = type::template get<double>();
03420     };
03421
03426     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>;
03429     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>;
03431     using SQRT2_fraction =
      ContinuedFraction<1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2>;
03433     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>;
      // NOLINT
03434 }  // namespace aerobus
```

```
03435
03436 // known polynomials
03437 namespace aerobus {
03438     // CChebyshev
03439     namespace internal {
03440         template<int kind, size_t deg, typename I>
03441         struct chebyshev_helper {
03442             using type = typename polynomial<I>::template sub_t<
03443                 typename polynomial<I>::template mul_t<
03444                     typename polynomial<I>::template mul_t<
03445                         typename polynomial<I>::template inject_constant_t<2>,
03446                         typename polynomial<I>::X>,
03447                     typename chebyshev_helper<kind, deg - 1, I>::type
03448                 >,
03449                 typename chebyshev_helper<kind, deg - 2, I>::type
03450             >;
03451         };
03452
03453         template<typename I>
03454         struct chebyshev_helper<1, 0, I> {
03455             using type = typename polynomial<I>::one;
03456         };
03457
03458         template<typename I>
03459         struct chebyshev_helper<1, 1, I> {
03460             using type = typename polynomial<I>::X;
03461         };
03462
03463         template<typename I>
03464         struct chebyshev_helper<2, 0, I> {
03465             using type = typename polynomial<I>::one;
03466         };
03467
03468         template<typename I>
03469         struct chebyshev_helper<2, 1, I> {
03470             using type = typename polynomial<I>::template mul_t<
03471                 typename polynomial<I>::template inject_constant_t<2>,
03472                 typename polynomial<I>::X>;
03473         };
03474     }  // namespace internal
03475
03476     // Laguerre
03477     namespace internal {
03478         template<size_t deg, typename I>
03479         struct laguerre_helper {
03480             using Q = FractionField<I>;
03481             using PQ = polynomial<Q>;
03482
03483          private:
03484             // Lk = (1 / k) * ((2 * k - 1 - x) * lkm1 - (k - 2)Lkm2)
03485             using lnm2 = typename laguerre_helper<deg - 2, I>::type;
03486             using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03487             // -x + 2k-1
03488             using p = typename PQ::template val<
03489                 typename Q::template inject_constant_t<-1>,
03490                 typename Q::template inject_constant_t<2 * deg - 1»;
03491             // 1/n
03492             using factor = typename PQ::template inject_ring_t<
03493                 typename Q::template
      val<typename I::one, typename I::template inject_constant_t<deg>»;
03494
03495          public:
03496             using type = typename PQ::template mul_t <
03497                 factor,
03498                 typename PQ::template sub_t<
03499                     typename PQ::template mul_t<
03500                         p,
03501                         lnm1
03502                     >,
03503                     typename PQ::template mul_t<
03504                         typename PQ::template inject_constant_t<deg-1>,
03505                         lnm2
03506                     >
03507                 >
03508             >;
03509         };
03510
03511         template<typename I>
03512         struct laguerre_helper<0, I> {
03513             using type = typename polynomial<FractionField<I>>::one;
03514         };
03515
03516         template<typename I>
03517         struct laguerre_helper<1, I> {
03518          private:
03519             using PQ = polynomial<FractionField<I>>;
03520          public:
```

```
03521             using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
03522         };
03523     }  // namespace internal
03524
03525     // Bernstein
03526     namespace internal {
03527         template<size_t i, size_t m, typename I, typename E = void>
03528         struct bernstein_helper {};
03529
03530         template<typename I>
03531         struct bernstein_helper<0, 0, I> {
03532             using type = typename polynomial<I>::one;
03533         };
03534
03535         template<size_t i, size_t m, typename I>
03536         struct bernstein_helper<i, m, I, std::enable_if_t<
03537                     (m > 0) && (i == 0)» {
03538          private:
03539             using P = polynomial<I>;
03540          public:
03541             using type = typename P::template mul_t<
03542                     typename P::template sub_t<typename P::one, typename P::X>,
03543                     typename bernstein_helper<i, m-1, I>::type>;
03544         };
03545
03546         template<size_t i, size_t m, typename I>
03547         struct bernstein_helper<i, m, I, std::enable_if_t<
03548                     (m > 0) && (i == m)» {
03549          private:
03550             using P = polynomial<I>;
03551          public:
03552             using type = typename P::template mul_t<
03553                     typename P::X,
03554                     typename bernstein_helper<i-1, m-1, I>::type>;
03555         };
03556
03557         template<size_t i, size_t m, typename I>
03558         struct bernstein_helper<i, m, I, std::enable_if_t<
03559                     (m > 0) && (i > 0) && (i < m)» {
03560          private:
03561             using P = polynomial<I>;
03562          public:
03563             using type = typename P::template add_t<
03564                     typename P::template mul_t<
03565                         typename P::template sub_t<typename P::one, typename P::X>,
03566                         typename bernstein_helper<i, m-1, I>::type>,
03567                     typename P::template mul_t<
03568                         typename P::X,
03569                         typename bernstein_helper<i-1, m-1, I>::type»;
03570         };
03571     }  // namespace internal
03572
03573     namespace known_polynomials {
03575         enum hermite_kind {
03577             probabilist,
03579             physicist
03580         };
03581     }
03582
03583     // hermite
03584     namespace internal {
03585         template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03586         struct hermite_helper {};
03587
03588         template<size_t deg, typename I>
03589         struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
03590          private:
03591             using hnm1 = typename hermite_helper<deg - 1,
      known_polynomials::hermite_kind::probabilist, I>::type;
03592             using hnm2 = typename hermite_helper<deg - 2,
      known_polynomials::hermite_kind::probabilist, I>::type;
03593
03594          public:
03595             using type = typename polynomial<I>::template sub_t<
03596                 typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnm1>,
03597                 typename polynomial<I>::template mul_t<
03598                     typename polynomial<I>::template inject_constant_t<deg - 1>,
03599                     hnm2
03600                 >
03601             >;
03602         };
03603
03604         template<size_t deg, typename I>
03605         struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
03606          private:
03607             using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
      I>::type;
```

```
03608             using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
      I>::type;
03609
03610          public:
03611             using type = typename polynomial<I>::template sub_t<
03612                 // 2X Hn-1
03613                 typename polynomial<I>::template mul_t<
03614                     typename pi64::val<typename I::template inject_constant_t<2>,
03615                     typename I::zero>, hnm1>,
03616
03617                 typename polynomial<I>::template mul_t<
03618                     typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
03619                     hnm2
03620                 >
03621             >;
03622         };
03623
03624         template<typename I>
03625         struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
03626             using type = typename polynomial<I>::one;
03627         };
03628
03629         template<typename I>
03630         struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
03631             using type = typename polynomial<I>::X;
03632         };
03633
03634         template<typename I>
03635         struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
03636             using type = typename pi64::one;
03637         };
03638
03639         template<typename I>
03640         struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
03641             // 2X
03642             using type = typename polynomial<I>::template val<
03643                 typename I::template inject_constant_t<2>,
03644                 typename I::zero>;
03645         };
03646     }  // namespace internal
03647
03648     // legendre
03649     namespace internal {
03650         template<size_t n, typename I>
03651         struct legendre_helper {
03652          private:
03653             using Q = FractionField<I>;
03654             using PQ = polynomial<Q>;
03655             // 1/n constant
03656             // (2n-1)/n X
03657             using fact_left = typename PQ::template monomial_t<
03658                 makefraction_t<I,
03659                     typename I::template inject_constant_t<2*n-1>,
03660                     typename I::template inject_constant_t<n>
03661                 >,
03662             1>;
03663             // (n-1) / n
03664             using fact_right = typename PQ::template val<
03665                 makefraction_t<I,
03666                     typename I::template inject_constant_t<n-1>,
03667                     typename I::template inject_constant_t<n>»;
03668
03669          public:
03670             using type = PQ::template sub_t<
03671                     typename PQ::template mul_t<
03672                         fact_left,
03673                         typename legendre_helper<n-1, I>::type
03674                     >,
03675                     typename PQ::template mul_t<
03676                         fact_right,
03677                         typename legendre_helper<n-2, I>::type
03678                     >
03679                 >;
03680         };
03681
03682         template<typename I>
03683         struct legendre_helper<0, I> {
03684             using type = typename polynomial<FractionField<I>>::one;
03685         };
03686
03687         template<typename I>
03688         struct legendre_helper<1, I> {
03689             using type = typename polynomial<FractionField<I>>::X;
03690         };
03691     }  // namespace internal
03692
03693     // bernoulli polynomials
```

```
03694     namespace internal {
03695         template<size_t n>
03696         struct bernoulli_coeff {
03697             template<typename T, size_t i>
03698             struct inner {
03699              private:
03700                 using F = FractionField<T>;
03701              public:
03702                 using type = typename F::template mul_t<
03703                     typename F::template inject_ring_t<combination_t<T, i, n>>,
03704                     bernoulli_t<T, n-i>
03705                 >;
03706             };
03707         };
03708     }  // namespace internal
03709
03711     namespace known_polynomials {
03719         template <size_t deg, typename I = aerobus::i64>
03720         using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
03721
03731         template <size_t deg, typename I = aerobus::i64>
03732         using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
03733
03743         template <size_t deg, typename I = aerobus::i64>
03744         using laguerre = typename internal::laguerre_helper<deg, I>::type;
03745
03752         template <size_t deg, typename I = aerobus::i64>
03753         using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
      I>::type;
03754
03761         template <size_t deg, typename I = aerobus::i64>
03762         using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
03763
03774         template<size_t i, size_t m, typename I = aerobus::i64>
03775         using bernstein = typename internal::bernstein_helper<i, m, I>::type;
03776
03786         template<size_t deg, typename I = aerobus::i64>
03787         using legendre = typename internal::legendre_helper<deg, I>::type;
03788
03798         template<size_t deg, typename I = aerobus::i64>
03799         using bernoulli = taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
03800     }  // namespace known_polynomials
03801 }  // namespace aerobus
03802
03803
03804 #ifdef AEROBUS_CONWAY_IMPORTS
03805
03806 // conway polynomials
03807 namespace aerobus {
03811     template<int p, int n>
03812     struct ConwayPolynomial {};
03813
03814 #ifndef DO_NOT_DOCUMENT
03815     #define ZPZV ZPZ::template val
03816     #define POLYV aerobus::polynomial<ZPZ>::template val
03817     template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<1>>; };  // NOLINT
03818     template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>>; }; // NOLINT
03819     template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>>; };  // NOLINT
03820     template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>>; };  // NOLINT
03821     template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>>; };  // NOLINT
03822     template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>>; };  // NOLINT
03823     template<> struct ConwayPolynomial<2, 7> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>>; };  // NOLINT
03824     template<> struct ConwayPolynomial<2, 8> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>>; };  // NOLINT
03825     template<> struct ConwayPolynomial<2, 9> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>>; };
      // NOLINT
03826     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>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>,
      ZPZV<1>>; };  // NOLINT
03827     template<> struct ConwayPolynomial<2, 11> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
      ZPZV<0>, ZPZV<1>>; };  // NOLINT
03828     template<> struct ConwayPolynomial<2, 12> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>,
      ZPZV<0>, ZPZV<1>, ZPZV<1>>; };  // NOLINT
03829     template<> struct ConwayPolynomial<2, 13> { 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<0>, ZPZV<1>, ZPZV<1>>; };  // NOLINT
03830     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<0>, ZPZV<1>,
      ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>>; };  // NOLINT
03831     template<> struct ConwayPolynomial<2, 15> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>>; };  // NOLINT
03832     template<> struct ConwayPolynomial<2, 16> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>>; };  // NOLINT
03833     template<> struct ConwayPolynomial<2, 17> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>>; };  // NOLINT
03834     template<> struct ConwayPolynomial<2, 18> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>>; };  // NOLINT
03835     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<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>>; };  //
      NOLINT
03836     template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>>; };
      // NOLINT
03837     template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<1>>; };  // NOLINT
03838     template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>>; };  // NOLINT
03839     template<> struct ConwayPolynomial<3, 3> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<1>>; };  // NOLINT
03840     template<> struct ConwayPolynomial<3, 4> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2>>; };  // NOLINT
03841     template<> struct ConwayPolynomial<3, 5> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>>; };  // NOLINT
03842     template<> struct ConwayPolynomial<3, 6> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<2>>; };  // NOLINT
03843     template<> struct ConwayPolynomial<3, 7> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>>; };  // NOLINT
03844     template<> struct ConwayPolynomial<3, 8> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>>; };  // NOLINT
03845     template<> struct ConwayPolynomial<3, 9> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1>>; };
      // NOLINT
03846     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
03847     template<> struct ConwayPolynomial<3, 11> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>,
      ZPZV<0>, ZPZV<1>>; };  // NOLINT
03848     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<0>,
      ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>>; };  // NOLINT
03849     template<> struct ConwayPolynomial<3, 13> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>>; };  // NOLINT
03850     template<> struct ConwayPolynomial<3, 14> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<1>,
      ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>>; };  // NOLINT
03851     template<> struct ConwayPolynomial<3, 15> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<0>,
      ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1>>; };  // NOLINT
03852     template<> struct ConwayPolynomial<3, 16> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>,
      ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2>>; };  // NOLINT
03853     template<> struct ConwayPolynomial<3, 17> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>>; };  // NOLINT
03854     template<> struct ConwayPolynomial<3, 18> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>,
      ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>>; };  // NOLINT
03855     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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>>; };  //
      NOLINT
03856     template<> struct ConwayPolynomial<3, 20> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>,
      ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<2>>; };
      // NOLINT
03857     template<> struct ConwayPolynomial<5, 1> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<3>>; };  // NOLINT
03858     template<> struct ConwayPolynomial<5, 2> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2>>; };  // NOLINT
03859     template<> struct ConwayPolynomial<5, 3> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3>>; };  // NOLINT
03860     template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2>>; };  // NOLINT
03861     template<> struct ConwayPolynomial<5, 5> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<3>>; };  // NOLINT
03862     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
03863     template<> struct ConwayPolynomial<5, 7> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>>; };  // NOLINT
03864     template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<2>>; };  // NOLINT
03865     template<> struct ConwayPolynomial<5, 9> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<3>>; };
      // NOLINT
03866     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<2>, ZPZV<4>, ZPZV<1>,
      ZPZV<2>>; };  // NOLINT
03867     template<> struct ConwayPolynomial<5, 11> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<3>, ZPZV<3>>; };  // NOLINT
03868     template<> struct ConwayPolynomial<5, 12> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>,
      ZPZV<3>, ZPZV<2>, ZPZV<2>>; };  // NOLINT
03869     template<> struct ConwayPolynomial<5, 13> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<3>>; };  // NOLINT
03870     template<> struct ConwayPolynomial<5, 14> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>,
      ZPZV<2>, ZPZV<3>, ZPZV<0>, ZPZV<1>, ZPZV<2>>; };  // NOLINT
03871     template<> struct ConwayPolynomial<5, 15> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<3>>; };  // NOLINT
03872     template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>,
      ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2>>; };  // NOLINT
03873     template<> struct ConwayPolynomial<5, 17> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<3>>; };  // NOLINT
03874     template<> struct ConwayPolynomial<5, 18> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>,
      ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<2>>; };  // NOLINT
03875     template<> struct ConwayPolynomial<5, 19> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<3>>; };  //
      NOLINT
03876     template<> struct ConwayPolynomial<5, 20> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<0>,
      ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<0>, ZPZV<1>, ZPZV<2>>; };
      // NOLINT
03877     template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<4>>; };  // NOLINT
03878     template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3>>; };  // NOLINT
03879     template<> struct ConwayPolynomial<7, 3> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4>>; };  // NOLINT
03880     template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3>>; };  // NOLINT
03881     template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>>; };  // NOLINT
03882     template<> struct ConwayPolynomial<7, 6> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<4>, ZPZV<6>, ZPZV<3>>; };  // NOLINT
03883     template<> struct ConwayPolynomial<7, 7> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<4>>; };  // NOLINT
03884     template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<6>, ZPZV<2>, ZPZV<3>>; };  // NOLINT
03885     template<> struct ConwayPolynomial<7, 9> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<4>>; };
      // NOLINT
03886     template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<3>,
      ZPZV<3>>; };  // NOLINT
03887     template<> struct ConwayPolynomial<7, 11> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<1>, ZPZV<4>>; };  // NOLINT
03888     template<> struct ConwayPolynomial<7, 12> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<5>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<0>,
      ZPZV<5>, ZPZV<0>, ZPZV<3>>; };  // NOLINT
03889     template<> struct ConwayPolynomial<7, 13> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<6>, ZPZV<0>, ZPZV<4>>; };  // NOLINT
03890     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<5>, ZPZV<0>, ZPZV<6>,
      ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<3>>; };  // NOLINT
03891     template<> struct ConwayPolynomial<7, 15> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>,
      ZPZV<6>, ZPZV<6>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<4>>; };  // NOLINT
03892     template<> struct ConwayPolynomial<7, 16> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<5>,
      ZPZV<3>, ZPZV<4>, ZPZV<1>, ZPZV<6>, ZPZV<2>, ZPZV<4>, ZPZV<3>>; };  // NOLINT
03893     template<> struct ConwayPolynomial<7, 17> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>>; };  // NOLINT
03894     template<> struct ConwayPolynomial<7, 18> { using ZPZ = aerobus::zpz<7>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<6>, ZPZV<1>,
      ZPZV<6>, ZPZV<5>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<2>, ZPZV<3>>>; };  // NOLINT
03895     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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<4>>>; };  //
      NOLINT
03896     template<> struct ConwayPolynomial<7, 20> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<6>,
      ZPZV<2>, ZPZV<5>, ZPZV<2>, ZPZV<3>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<1>, ZPZV<3>>>; };
      // NOLINT
03897     template<> struct ConwayPolynomial<11, 1> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<9>>; };  // NOLINT
03898     template<> struct ConwayPolynomial<11, 2> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2>>; };  // NOLINT
03899     template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<9>>; };  // NOLINT
03900     template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2>>; };  // NOLINT
03901     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
03902     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
03903     template<> struct ConwayPolynomial<11, 7> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<9>>>; };  // NOLINT
03904     template<> struct ConwayPolynomial<11, 8> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<1>, ZPZV<7>, ZPZV<2>>>; };  // NOLINT
03905     template<> struct ConwayPolynomial<11, 9> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<8>, ZPZV<9>>>; };
      // NOLINT
03906     template<> struct ConwayPolynomial<11, 10> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<10>, ZPZV<6>, ZPZV<6>,
      ZPZV<2>>>; };  // NOLINT
03907     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<10>, ZPZV<9>>>; };  // NOLINT
03908     template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<2>, ZPZV<5>, ZPZV<5>,
      ZPZV<6>, ZPZV<5>, ZPZV<2>>>; };  // NOLINT
03909     template<> struct ConwayPolynomial<11, 13> { 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<7>, ZPZV<9>>>; };  // NOLINT
03910     template<> struct ConwayPolynomial<11, 14> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<9>, ZPZV<6>,
      ZPZV<4>, ZPZV<8>, ZPZV<6>, ZPZV<10>, ZPZV<2>>>; };  // NOLINT
03911     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<10>,
      ZPZV<7>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<0>, ZPZV<9>>>; };  // NOLINT
03912     template<> struct ConwayPolynomial<11, 16> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>,
      ZPZV<1>, ZPZV<3>, ZPZV<5>, ZPZV<3>, ZPZV<10>, ZPZV<9>, ZPZV<2>>>; };  // NOLINT
03913     template<> struct ConwayPolynomial<11, 17> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<9>>>; };  // NOLINT
03914     template<> struct ConwayPolynomial<11, 18> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<10>, ZPZV<8>,
      ZPZV<3>, ZPZV<9>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<9>, ZPZV<8>, ZPZV<2>, ZPZV<2>>>; };  // NOLINT
03915     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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<2>, ZPZV<9>>>; };  //
      NOLINT
03916     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<1>, ZPZV<10>,
      ZPZV<9>, ZPZV<1>, ZPZV<5>, ZPZV<7>, ZPZV<2>, ZPZV<4>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZPZV<2>>>; };
      // NOLINT
03917     template<> struct ConwayPolynomial<13, 1> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<11>>; };  // NOLINT
03918     template<> struct ConwayPolynomial<13, 2> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<12>, ZPZV<2>>>; };  // NOLINT
03919     template<> struct ConwayPolynomial<13, 3> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>>>; };  // NOLINT
03920     template<> struct ConwayPolynomial<13, 4> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2>>>; };  // NOLINT
03921     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
03922     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
03923     template<> struct ConwayPolynomial<13, 7> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<11>>>; };  // NOLINT
03924     template<> struct ConwayPolynomial<13, 8> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<12>, ZPZV<2>, ZPZV<3>, ZPZV<2>>>; };  // NOLINT
03925     template<> struct ConwayPolynomial<13, 9> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<8>, ZPZV<12>, ZPZV<12>, ZPZV<11>>>;
      };  // NOLINT
03926     template<> struct ConwayPolynomial<13, 10> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<5>, ZPZV<8>, ZPZV<1>, ZPZV<1>,
      ZPZV<2>>>; };  // NOLINT
03927     template<> struct ConwayPolynomial<13, 11> { 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<3>, ZPZV<11>>; };  // NOLINT
03928     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<1>, ZPZV<4>, ZPZV<2>>; };  // NOLINT
03929     template<> struct ConwayPolynomial<13, 13> { 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<0>, ZPZV<0>, ZPZV<12>, ZPZV<11>>; };  // NOLINT
03930     template<> struct ConwayPolynomial<13, 14> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<0>, ZPZV<6>,
      ZPZV<11>, ZPZV<7>, ZPZV<10>, ZPZV<10>, ZPZV<2>>; };  // NOLINT
03931     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<2>, ZPZV<12>,
      ZPZV<2>, ZPZV<11>, ZPZV<10>, ZPZV<11>, ZPZV<8>, ZPZV<11>>; };  // NOLINT
03932     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<3>, ZPZV<12>,
      ZPZV<8>, ZPZV<2>, ZPZV<12>, ZPZV<9>, ZPZV<12>, ZPZV<6>, ZPZV<2>>; };  // NOLINT
03933     template<> struct ConwayPolynomial<13, 17> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<6>, ZPZV<11>>; };  // NOLINT
03934     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<10>, ZPZV<4>, ZPZV<11>,
      ZPZV<11>, ZPZV<9>, ZPZV<5>, ZPZV<3>, ZPZV<5>, ZPZV<6>, ZPZV<0>, ZPZV<9>, ZPZV<2>>; };  // NOLINT
03935     template<> struct ConwayPolynomial<13, 19> { 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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<11>>; };  //
      NOLINT
03936     template<> struct ConwayPolynomial<13, 20> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>,
      ZPZV<9>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<7>, ZPZV<4>, ZPZV<0>, ZPZV<4>, ZPZV<8>, ZPZV<11>, ZPZV<2>>;
      };  // NOLINT
03937     template<> struct ConwayPolynomial<17, 1> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<14>>; };  // NOLINT
03938     template<> struct ConwayPolynomial<17, 2> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3>>; };  // NOLINT
03939     template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14>>; };  // NOLINT
03940     template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<10>, ZPZV<3>>; };  // NOLINT
03941     template<> struct ConwayPolynomial<17, 5> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14>>; };  // NOLINT
03942     template<> struct ConwayPolynomial<17, 6> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<10>, ZPZV<3>, ZPZV<3>>; };  // NOLINT
03943     template<> struct ConwayPolynomial<17, 7> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<14>>; };  // NOLINT
03944     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
03945     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<14>>; };
      // NOLINT
03946     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<9>, ZPZV<12>,
      ZPZV<3>>; };  // NOLINT
03947     template<> struct ConwayPolynomial<17, 11> { 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<14>>; };  // NOLINT
03948     template<> struct ConwayPolynomial<17, 12> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<14>, ZPZV<14>, ZPZV<13>, ZPZV<6>,
      ZPZV<14>, ZPZV<9>, ZPZV<3>>; };  // NOLINT
03949     template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<14>>; };  // NOLINT
03950     template<> struct ConwayPolynomial<17, 14> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<1>, ZPZV<8>,
      ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3>, ZPZV<3>>; };  // NOLINT
03951     template<> struct ConwayPolynomial<17, 15> { 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<4>,
      ZPZV<4>, ZPZV<16>, ZPZV<6>, ZPZV<14>, ZPZV<14>, ZPZV<14>>; };  // NOLINT
03952     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<1>, ZPZV<13>,
      ZPZV<5>, ZPZV<2>, ZPZV<12>, ZPZV<13>, ZPZV<12>, ZPZV<1>, ZPZV<3>>; };  // NOLINT
03953     template<> struct ConwayPolynomial<17, 17> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<14>>; };  // NOLINT
03954     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<0>, ZPZV<9>, ZPZV<16>,
      ZPZV<7>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<11>, ZPZV<13>, ZPZV<13>, ZPZV<9>, ZPZV<3>>; };  // NOLINT
03955     template<> struct ConwayPolynomial<17, 19> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<14>>; };  //
      NOLINT
03956     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<1>, 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
03957     template<> struct ConwayPolynomial<19, 1> { using ZPZ = aerobus::zpz<19>; using type =
```

```
    POLYV<ZPZV<1>, ZPZV<17>>; };  // NOLINT
03958     template<> struct ConwayPolynomial<19, 2> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2>>; };  // NOLINT
03959     template<> struct ConwayPolynomial<19, 3> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17>>; };  // NOLINT
03960     template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2>>; };  // NOLINT
03961     template<> struct ConwayPolynomial<19, 5> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17>>; };  // NOLINT
03962     template<> struct ConwayPolynomial<19, 6> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<17>, ZPZV<6>, ZPZV<2>>; };  // NOLINT
03963     template<> struct ConwayPolynomial<19, 7> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<17>>; };  // NOLINT
03964     template<> struct ConwayPolynomial<19, 8> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<10>, ZPZV<3>, ZPZV<2>>; };  //
    NOLINT
03965     template<> struct ConwayPolynomial<19, 9> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<14>, ZPZV<16>, ZPZV<17>>;
    };  // NOLINT
03966     template<> struct ConwayPolynomial<19, 10> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<13>, ZPZV<17>, ZPZV<3>, ZPZV<4>,
    ZPZV<2>>; };  // NOLINT
03967     template<> struct ConwayPolynomial<19, 11> { 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<8>, ZPZV<17>>; };  // NOLINT
03968     template<> struct ConwayPolynomial<19, 12> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<18>, ZPZV<2>, ZPZV<9>,
    ZPZV<16>, ZPZV<7>, ZPZV<2>>; };  // NOLINT
03969     template<> struct ConwayPolynomial<19, 13> { 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<11>, ZPZV<17>>; };  // NOLINT
03970     template<> struct ConwayPolynomial<19, 14> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<11>, ZPZV<11>,
    ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<7>, ZPZV<2>>; };  // NOLINT
03971     template<> struct ConwayPolynomial<19, 15> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>,
    ZPZV<11>, ZPZV<13>, ZPZV<15>, ZPZV<14>, ZPZV<0>, ZPZV<17>>; };  // NOLINT
03972     template<> struct ConwayPolynomial<19, 16> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>,
    ZPZV<13>, ZPZV<0>, ZPZV<15>, ZPZV<9>, ZPZV<6>, ZPZV<14>, ZPZV<2>>; };  // NOLINT
03973     template<> struct ConwayPolynomial<19, 17> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<17>>; };  // NOLINT
03974     template<> struct ConwayPolynomial<19, 18> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<9>, ZPZV<7>,
    ZPZV<17>, ZPZV<5>, ZPZV<0>, ZPZV<16>, ZPZV<5>, ZPZV<7>, ZPZV<3>, ZPZV<14>, ZPZV<2>>; };  // NOLINT
03975     template<> struct ConwayPolynomial<19, 19> { using ZPZ = aerobus::zpz<19>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<17>>; };  //
    NOLINT
03976     template<> struct ConwayPolynomial<19, 20> { 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<16>,
    ZPZV<13>, ZPZV<0>, ZPZV<4>, ZPZV<7>, ZPZV<8>, ZPZV<6>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<11>, ZPZV<2>>;
    };  // NOLINT
03977     template<> struct ConwayPolynomial<23, 1> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<18>>; };  // NOLINT
03978     template<> struct ConwayPolynomial<23, 2> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5>>; };  // NOLINT
03979     template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18>>; };  // NOLINT
03980     template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5>>; };  // NOLINT
03981     template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<18>>; };  // NOLINT
03982     template<> struct ConwayPolynomial<23, 6> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<9>, ZPZV<1>, ZPZV<5>>; };  // NOLINT
03983     template<> struct ConwayPolynomial<23, 7> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<18>>; };  // NOLINT
03984     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<3>, ZPZV<5>>; };  // NOLINT
03985     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
03986     template<> struct ConwayPolynomial<23, 10> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<5>, ZPZV<15>, ZPZV<6>, ZPZV<1>,
    ZPZV<5>>; };  // NOLINT
03987     template<> struct ConwayPolynomial<23, 11> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>,
    ZPZV<7>, ZPZV<18>>; };  // NOLINT
03988     template<> struct ConwayPolynomial<23, 12> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<21>, ZPZV<15>, ZPZV<14>, ZPZV<12>,
    ZPZV<18>, ZPZV<12>, ZPZV<5>>; };  // NOLINT
03989     template<> struct ConwayPolynomial<23, 13> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<18>>; };  // NOLINT
03990     template<> struct ConwayPolynomial<23, 14> { using ZPZ = aerobus::zpz<23>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<1>,
```

```
      ZPZV<18>, ZPZV<19>, ZPZV<1>, ZPZV<22>, ZPZV<5>>; };  // NOLINT
03991     template<> struct ConwayPolynomial<23, 15> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>,
      ZPZV<8>, ZPZV<15>, ZPZV<9>, ZPZV<7>, ZPZV<18>, ZPZV<18>>; };  // NOLINT
03992     template<> struct ConwayPolynomial<23, 16> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>,
      ZPZV<19>, ZPZV<16>, ZPZV<13>, ZPZV<1>, ZPZV<14>, ZPZV<17>, ZPZV<5>>; };  // NOLINT
03993     template<> struct ConwayPolynomial<23, 17> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<18>>; };  // NOLINT
03994     template<> struct ConwayPolynomial<23, 18> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<18>, ZPZV<2>, ZPZV<1>,
      ZPZV<18>, ZPZV<3>, ZPZV<16>, ZPZV<21>, ZPZV<0>, ZPZV<11>, ZPZV<3>, ZPZV<19>, ZPZV<5>>; };  // NOLINT
03995     template<> struct ConwayPolynomial<23, 19> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<18>>; };  //
      NOLINT
03996     template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<27>>; };  // NOLINT
03997     template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2>>; };  // NOLINT
03998     template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<27>>; };  // NOLINT
03999     template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<2>>; };  // NOLINT
04000     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
04001     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
04002     template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<27>>; };  // NOLINT
04003     template<> struct ConwayPolynomial<29, 8> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<2>>; };  //
      NOLINT
04004     template<> struct ConwayPolynomial<29, 9> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<22>, ZPZV<22>, ZPZV<27>>; };
      // NOLINT
04005     template<> struct ConwayPolynomial<29, 10> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<8>, ZPZV<17>, ZPZV<2>, ZPZV<22>,
      ZPZV<2>>; };  // NOLINT
04006     template<> struct ConwayPolynomial<29, 11> { 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<28>,
      ZPZV<8>, ZPZV<27>>; };  // NOLINT
04007     template<> struct ConwayPolynomial<29, 12> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<28>, ZPZV<9>, ZPZV<16>, ZPZV<25>,
      ZPZV<1>, ZPZV<1>, ZPZV<2>>; };  // NOLINT
04008     template<> struct ConwayPolynomial<29, 13> { 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<7>, ZPZV<27>>; };  // NOLINT
04009     template<> struct ConwayPolynomial<29, 14> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<14>, ZPZV<10>,
      ZPZV<21>, ZPZV<18>, ZPZV<27>, ZPZV<5>, ZPZV<2>>; };  // NOLINT
04010     template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>,
      ZPZV<14>, ZPZV<8>, ZPZV<1>, ZPZV<12>, ZPZV<26>, ZPZV<27>>; };  // NOLINT
04011     template<> struct ConwayPolynomial<29, 16> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<27>,
      ZPZV<2>, ZPZV<18>, ZPZV<23>, ZPZV<1>, ZPZV<27>, ZPZV<10>, ZPZV<2>>; };  // NOLINT
04012     template<> struct ConwayPolynomial<29, 17> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<27>>; };  // NOLINT
04013     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<0>, ZPZV<24>, ZPZV<1>, ZPZV<1>,
      ZPZV<6>, ZPZV<26>, ZPZV<2>, ZPZV<10>, ZPZV<8>, ZPZV<16>, ZPZV<19>, ZPZV<14>, ZPZV<2>>; };  // NOLINT
04014     template<> struct ConwayPolynomial<29, 19> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<27>>; };  //
      NOLINT
04015     template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<28>>; };  // NOLINT
04016     template<> struct ConwayPolynomial<31, 2> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<29>, ZPZV<3>>; };  // NOLINT
04017     template<> struct ConwayPolynomial<31, 3> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<28>>; };  // NOLINT
04018     template<> struct ConwayPolynomial<31, 4> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3>>; };  // NOLINT
04019     template<> struct ConwayPolynomial<31, 5> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28>>; };  // NOLINT
04020     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
04021     template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28>>; };  // NOLINT
04022     template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<12>, ZPZV<24>, ZPZV<3>>; };  //
      NOLINT
04023     template<> struct ConwayPolynomial<31, 9> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<20>, ZPZV<29>, ZPZV<28>>; };
```

```
    // NOLINT
04024     template<> struct ConwayPolynomial<31, 10> { using ZPZ = aerobus::zpz<31>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<26>, ZPZV<13>, ZPZV<13>, ZPZV<13>,
    ZPZV<3>>; };  // NOLINT
04025     template<> struct ConwayPolynomial<31, 11> { 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<20>, ZPZV<28>>; };  // NOLINT
04026     template<> struct ConwayPolynomial<31, 12> { using ZPZ = aerobus::zpz<31>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<14>, ZPZV<28>, ZPZV<2>, ZPZV<9>,
    ZPZV<25>, ZPZV<12>, ZPZV<3>>; };  // NOLINT
04027     template<> struct ConwayPolynomial<31, 13> { 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<6>, ZPZV<28>>; };  // NOLINT
04028     template<> struct ConwayPolynomial<31, 14> { using ZPZ = aerobus::zpz<31>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<5>, ZPZV<1>,
    ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<6>, ZPZV<3>>; };  // NOLINT
04029     template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>,
    ZPZV<29>, ZPZV<12>, ZPZV<13>, ZPZV<23>, ZPZV<25>, ZPZV<28>>; };  // NOLINT
04030     template<> struct ConwayPolynomial<31, 16> { 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<28>,
    ZPZV<24>, ZPZV<26>, ZPZV<28>, ZPZV<11>, ZPZV<19>, ZPZV<27>, ZPZV<3>>; };  // NOLINT
04031     template<> struct ConwayPolynomial<31, 17> { using ZPZ = aerobus::zpz<31>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<28>>; };  // NOLINT
04032     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<1>, ZPZV<27>, ZPZV<5>, ZPZV<24>,
    ZPZV<2>, ZPZV<7>, ZPZV<12>, ZPZV<11>, ZPZV<25>, ZPZV<25>, ZPZV<10>, ZPZV<6>, ZPZV<3>>; };  // NOLINT
04033     template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28>>; };  //
    NOLINT
04034     template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<35>>; };  // NOLINT
04035     template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2>>; };  // NOLINT
04036     template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35>>; };  // NOLINT
04037     template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2>>; };  // NOLINT
04038     template<> struct ConwayPolynomial<37, 5> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35>>; };  // NOLINT
04039     template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<4>, ZPZV<30>, ZPZV<2>>; };  // NOLINT
04040     template<> struct ConwayPolynomial<37, 7> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<35>>; };  // NOLINT
04041     template<> struct ConwayPolynomial<37, 8> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<20>, ZPZV<27>, ZPZV<1>, ZPZV<2>>; };  //
    NOLINT
04042     template<> struct ConwayPolynomial<37, 9> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<20>, ZPZV<32>, ZPZV<35>>; };
    // NOLINT
04043     template<> struct ConwayPolynomial<37, 10> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<29>, ZPZV<18>, ZPZV<11>, ZPZV<4>,
    ZPZV<2>>; };  // NOLINT
04044     template<> struct ConwayPolynomial<37, 11> { 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<2>, ZPZV<35>>; };  // NOLINT
04045     template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<10>, ZPZV<23>, ZPZV<23>,
    ZPZV<18>, ZPZV<33>, ZPZV<2>>; };  // NOLINT
04046     template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35>>; };  // NOLINT
04047     template<> struct ConwayPolynomial<37, 14> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35>, ZPZV<35>, ZPZV<1>,
    ZPZV<32>, ZPZV<16>, ZPZV<1>, ZPZV<9>, ZPZV<2>>; };  // NOLINT
04048     template<> struct ConwayPolynomial<37, 15> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<31>,
    ZPZV<28>, ZPZV<27>, ZPZV<13>, ZPZV<34>, ZPZV<33>, ZPZV<35>>; };  // NOLINT
04049     template<> struct ConwayPolynomial<37, 17> { using ZPZ = aerobus::zpz<37>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>>; };  // NOLINT
04050     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<8>, ZPZV<19>, ZPZV<15>,
    ZPZV<1>, ZPZV<22>, ZPZV<20>, ZPZV<12>, ZPZV<32>, ZPZV<14>, ZPZV<27>, ZPZV<20>, ZPZV<2>>; };  // NOLINT
04051     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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<23>, ZPZV<35>>; };  //
    NOLINT
04052     template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<35>>; };  // NOLINT
04053     template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6>>; };  // NOLINT
04054     template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<35>>; };  // NOLINT
04055     template<> struct ConwayPolynomial<41, 4> { using ZPZ = aerobus::zpz<41>; using type =
```

```
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<6>>; };  // NOLINT
04056     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
04057     template<> struct ConwayPolynomial<41, 6> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<33>, ZPZV<39>, ZPZV<6>, ZPZV<6>>; };  // NOLINT
04058     template<> struct ConwayPolynomial<41, 7> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35>>; };  // NOLINT
04059     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>>; };  //
    NOLINT
04060     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
04061     template<> struct ConwayPolynomial<41, 10> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<8>, ZPZV<20>, ZPZV<30>,
    ZPZV<6>>; };  // NOLINT
04062     template<> struct ConwayPolynomial<41, 11> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<20>, ZPZV<35>>; };  // NOLINT
04063     template<> struct ConwayPolynomial<41, 12> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<13>, ZPZV<34>, ZPZV<24>,
    ZPZV<21>, ZPZV<27>, ZPZV<6>>; };  // NOLINT
04064     template<> struct ConwayPolynomial<41, 13> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<35>>; };  // NOLINT
04065     template<> struct ConwayPolynomial<41, 14> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<15>, ZPZV<4>,
    ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6>>; };  // NOLINT
04066     template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>,
    ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35>>; };  // NOLINT
04067     template<> struct ConwayPolynomial<41, 17> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35>>; };  // NOLINT
04068     template<> struct ConwayPolynomial<41, 18> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<20>,
    ZPZV<23>, ZPZV<35>, ZPZV<38>, ZPZV<24>, ZPZV<12>, ZPZV<29>, ZPZV<10>, ZPZV<6>, ZPZV<6>>; };  // NOLINT
04069     template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35>>; };  //
    NOLINT
04070     template<> struct ConwayPolynomial<43, 1> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<40>>; };  // NOLINT
04071     template<> struct ConwayPolynomial<43, 2> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3>>; };  // NOLINT
04072     template<> struct ConwayPolynomial<43, 3> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40>>; };  // NOLINT
04073     template<> struct ConwayPolynomial<43, 4> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3>>; };  // NOLINT
04074     template<> struct ConwayPolynomial<43, 5> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<40>>; };  // NOLINT
04075     template<> struct ConwayPolynomial<43, 6> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<21>, ZPZV<3>>; };  // NOLINT
04076     template<> struct ConwayPolynomial<43, 7> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<7>, ZPZV<40>>; };  // NOLINT
04077     template<> struct ConwayPolynomial<43, 8> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<39>, ZPZV<20>, ZPZV<24>, ZPZV<3>>; };  //
    NOLINT
04078     template<> struct ConwayPolynomial<43, 9> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<39>, ZPZV<1>, ZPZV<40>>; };
    // NOLINT
04079     template<> struct ConwayPolynomial<43, 10> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<26>, ZPZV<36>, ZPZV<5>, ZPZV<27>, ZPZV<24>,
    ZPZV<3>>; };  // NOLINT
04080     template<> struct ConwayPolynomial<43, 11> { 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<7>, ZPZV<40>>; };  // NOLINT
04081     template<> struct ConwayPolynomial<43, 12> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<27>, ZPZV<16>, ZPZV<17>, ZPZV<6>,
    ZPZV<23>, ZPZV<38>, ZPZV<3>>; };  // NOLINT
04082     template<> struct ConwayPolynomial<43, 13> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<40>>; };  // NOLINT
04083     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<38>, ZPZV<22>, ZPZV<24>,
    ZPZV<37>, ZPZV<18>, ZPZV<4>, ZPZV<19>, ZPZV<3>>; };  // NOLINT
04084     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<2>, ZPZV<37>,
    ZPZV<22>, ZPZV<42>, ZPZV<4>, ZPZV<15>, ZPZV<37>, ZPZV<40>>; };  // NOLINT
04085     template<> struct ConwayPolynomial<43, 17> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<40>>; };  // NOLINT
04086     template<> struct ConwayPolynomial<43, 18> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<28>, ZPZV<41>,
    ZPZV<24>, ZPZV<7>, ZPZV<24>, ZPZV<29>, ZPZV<16>, ZPZV<34>, ZPZV<37>, ZPZV<18>, ZPZV<3>>; };  // NOLINT
04087     template<> struct ConwayPolynomial<43, 19> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
```

```
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<40>>; };  //
      NOLINT
04088     template<> struct ConwayPolynomial<47, 1> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<42>>; };  // NOLINT
04089     template<> struct ConwayPolynomial<47, 2> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<45>, ZPZV<5>>; };  // NOLINT
04090     template<> struct ConwayPolynomial<47, 3> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42>>; };  // NOLINT
04091     template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5>>; };  // NOLINT
04092     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
04093     template<> struct ConwayPolynomial<47, 6> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<35>, ZPZV<9>, ZPZV<41>, ZPZV<5>>; };  // NOLINT
04094     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<12>, ZPZV<42>>; };  // NOLINT
04095     template<> struct ConwayPolynomial<47, 8> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<19>, ZPZV<3>, ZPZV<5>>; };  //
      NOLINT
04096     template<> struct ConwayPolynomial<47, 9> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<19>, ZPZV<1>, ZPZV<42>>; };
      // NOLINT
04097     template<> struct ConwayPolynomial<47, 10> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42>, ZPZV<14>, ZPZV<18>, ZPZV<45>, ZPZV<45>,
      ZPZV<5>>; };  // NOLINT
04098     template<> struct ConwayPolynomial<47, 11> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<6>, ZPZV<42>>; };  // NOLINT
04099     template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<40>, ZPZV<35>, ZPZV<12>, ZPZV<46>,
      ZPZV<14>, ZPZV<9>, ZPZV<5>>; };  // NOLINT
04100     template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<42>>; };  // NOLINT
04101     template<> struct ConwayPolynomial<47, 14> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<20>, ZPZV<30>,
      ZPZV<17>, ZPZV<24>, ZPZV<9>, ZPZV<32>, ZPZV<5>>; };  // NOLINT
04102     template<> struct ConwayPolynomial<47, 15> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<43>,
      ZPZV<31>, ZPZV<14>, ZPZV<42>, ZPZV<13>, ZPZV<17>, ZPZV<42>>; };  // NOLINT
04103     template<> struct ConwayPolynomial<47, 17> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<42>>; };  // NOLINT
04104     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<41>, ZPZV<42>,
      ZPZV<26>, ZPZV<44>, ZPZV<24>, ZPZV<22>, ZPZV<11>, ZPZV<5>, ZPZV<45>, ZPZV<33>, ZPZV<5>>; };  // NOLINT
04105     template<> struct ConwayPolynomial<47, 19> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<42>>; };  //
      NOLINT
04106     template<> struct ConwayPolynomial<53, 1> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<51>>; };  // NOLINT
04107     template<> struct ConwayPolynomial<53, 2> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2>>; };  // NOLINT
04108     template<> struct ConwayPolynomial<53, 3> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51>>; };  // NOLINT
04109     template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<38>, ZPZV<2>>; };  // NOLINT
04110     template<> struct ConwayPolynomial<53, 5> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<51>>; };  // NOLINT
04111     template<> struct ConwayPolynomial<53, 6> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<4>, ZPZV<45>, ZPZV<2>>; };  // NOLINT
04112     template<> struct ConwayPolynomial<53, 7> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<51>>; };  // NOLINT
04113     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>>; };  //
      NOLINT
04114     template<> struct ConwayPolynomial<53, 9> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<5>, ZPZV<51>>; };
      // NOLINT
04115     template<> struct ConwayPolynomial<53, 10> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<15>, ZPZV<29>,
      ZPZV<2>>; };  // NOLINT
04116     template<> struct ConwayPolynomial<53, 11> { 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<15>, ZPZV<51>>; };  // NOLINT
04117     template<> struct ConwayPolynomial<53, 12> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<34>, ZPZV<4>, ZPZV<13>, ZPZV<10>, ZPZV<42>,
      ZPZV<34>, ZPZV<41>, ZPZV<2>>; };  // NOLINT
04118     template<> struct ConwayPolynomial<53, 13> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<52>, ZPZV<28>, ZPZV<51>>; };  // NOLINT
04119     template<> struct ConwayPolynomial<53, 14> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<45>, ZPZV<23>, ZPZV<52>,
      ZPZV<0>, ZPZV<37>, ZPZV<12>, ZPZV<23>, ZPZV<2>>; };  // NOLINT
04120     template<> struct ConwayPolynomial<53, 15> { 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<22>,
```

```
      ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51>>; };  // NOLINT
04121     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<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<51>>; };  // NOLINT
04122     template<> struct ConwayPolynomial<53, 18> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<52>, ZPZV<31>, ZPZV<51>,
      ZPZV<27>, ZPZV<0>, ZPZV<39>, ZPZV<44>, ZPZV<6>, ZPZV<8>, ZPZV<16>, ZPZV<11>, ZPZV<2>>; };  // NOLINT
04123     template<> struct ConwayPolynomial<53, 19> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<51>>; };  //
      NOLINT
04124     template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<57>>; };  // NOLINT
04125     template<> struct ConwayPolynomial<59, 2> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2>>; };  // NOLINT
04126     template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57>>; };  // NOLINT
04127     template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<2>>; };  // NOLINT
04128     template<> struct ConwayPolynomial<59, 5> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<57>>; };  // NOLINT
04129     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
04130     template<> struct ConwayPolynomial<59, 7> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<57>>; };  // NOLINT
04131     template<> struct ConwayPolynomial<59, 8> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<32>, ZPZV<2>, ZPZV<50>, ZPZV<2>>; };  //
      NOLINT
04132     template<> struct ConwayPolynomial<59, 9> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<32>, ZPZV<47>, ZPZV<57>>; };
      // NOLINT
04133     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
04134     template<> struct ConwayPolynomial<59, 11> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<6>, ZPZV<57>>; };  // NOLINT
04135     template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<21>, ZPZV<38>,
      ZPZV<8>, ZPZV<1>, ZPZV<2>>; };  // NOLINT
04136     template<> struct ConwayPolynomial<59, 13> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57>>; };  // NOLINT
04137     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
04138     template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>,
      ZPZV<24>, ZPZV<23>, ZPZV<13>, ZPZV<39>, ZPZV<58>, ZPZV<57>>; };  // NOLINT
04139     template<> struct ConwayPolynomial<59, 17> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<57>>; };  // NOLINT
04140     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<1>, ZPZV<37>, ZPZV<38>, ZPZV<27>,
      ZPZV<11>, ZPZV<14>, ZPZV<7>, ZPZV<44>, ZPZV<16>, ZPZV<47>, ZPZV<34>, ZPZV<32>, ZPZV<2>>; };  // NOLINT
04141     template<> struct ConwayPolynomial<59, 19> { using ZPZ = aerobus::zpz<59>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<57>>; };  //
      NOLINT
04142     template<> struct ConwayPolynomial<61, 1> { using ZPZ = aerobus::zpz<61>; using type =
      POLYV<ZPZV<1>, ZPZV<59>>; };  // NOLINT
04143     template<> struct ConwayPolynomial<61, 2> { using ZPZ = aerobus::zpz<61>; using type =
      POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2>>; };  // NOLINT
04144     template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59>>; };  // NOLINT
04145     template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2>>; };  // NOLINT
04146     template<> struct ConwayPolynomial<61, 5> { using ZPZ = aerobus::zpz<61>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59>>; };  // NOLINT
04147     template<> struct ConwayPolynomial<61, 6> { using ZPZ = aerobus::zpz<61>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<3>, ZPZV<29>, ZPZV<2>>; };  // NOLINT
04148     template<> struct ConwayPolynomial<61, 7> { using ZPZ = aerobus::zpz<61>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59>>; };  // NOLINT
04149     template<> struct ConwayPolynomial<61, 8> { using ZPZ = aerobus::zpz<61>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<1>, ZPZV<56>, ZPZV<2>>; };  //
      NOLINT
04150     template<> struct ConwayPolynomial<61, 9> { using ZPZ = aerobus::zpz<61>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<50>, ZPZV<18>, ZPZV<59>>; };
      // NOLINT
04151     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
04152     template<> struct ConwayPolynomial<61, 11> { 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<18>, ZPZV<59>>; };  // NOLINT
04153     template<> struct ConwayPolynomial<61, 12> { using ZPZ = aerobus::zpz<61>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<42>, ZPZV<33>, ZPZV<8>, ZPZV<38>, ZPZV<14>,
```

```
ZPZV<1>, ZPZV<15>, ZPZV<2>>; };  // NOLINT
04154     template<> struct ConwayPolynomial<61, 13> { 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<3>, ZPZV<59>>; };  // NOLINT
04155     template<> struct ConwayPolynomial<61, 14> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, 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
04156     template<> struct ConwayPolynomial<61, 15> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>,
ZPZV<35>, ZPZV<44>, ZPZV<25>, ZPZV<23>, ZPZV<51>, ZPZV<59>>; };  // NOLINT
04157     template<> struct ConwayPolynomial<61, 17> { 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<10>, ZPZV<59>>; };  // NOLINT
04158     template<> struct ConwayPolynomial<61, 18> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<36>, ZPZV<13>,
ZPZV<36>, ZPZV<4>, ZPZV<32>, ZPZV<57>, ZPZV<42>, ZPZV<25>, ZPZV<25>, ZPZV<52>, ZPZV<2>>; };  // NOLINT
04159     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<2>, ZPZV<59>>; };  //
NOLINT
04160     template<> struct ConwayPolynomial<67, 1> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<65>>; };  // NOLINT
04161     template<> struct ConwayPolynomial<67, 2> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2>>; };  // NOLINT
04162     template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65>>; };  // NOLINT
04163     template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<54>, ZPZV<2>>; };  // NOLINT
04164     template<> struct ConwayPolynomial<67, 5> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<65>>; };  // NOLINT
04165     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
04166     template<> struct ConwayPolynomial<67, 7> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<65>>; };  // NOLINT
04167     template<> struct ConwayPolynomial<67, 8> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<46>, ZPZV<17>, ZPZV<64>, ZPZV<2>>; };  //
NOLINT
04168     template<> struct ConwayPolynomial<67, 9> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<49>, ZPZV<55>, ZPZV<65>>;
};  // NOLINT
04169     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
04170     template<> struct ConwayPolynomial<67, 11> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>,
ZPZV<9>, ZPZV<65>>; };  // NOLINT
04171     template<> struct ConwayPolynomial<67, 12> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<57>, ZPZV<27>, ZPZV<4>, ZPZV<55>, ZPZV<64>,
ZPZV<21>, ZPZV<27>, ZPZV<2>>; };  // NOLINT
04172     template<> struct ConwayPolynomial<67, 13> { 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<22>, ZPZV<65>>; };  // NOLINT
04173     template<> struct ConwayPolynomial<67, 14> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<22>, ZPZV<5>,
ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<2>>; };  // NOLINT
04174     template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>,
ZPZV<52>, ZPZV<41>, ZPZV<20>, ZPZV<21>, ZPZV<46>, ZPZV<65>>; };  // NOLINT
04175     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<5>, ZPZV<65>>; };  // NOLINT
04176     template<> struct ConwayPolynomial<67, 18> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<63>, ZPZV<52>, ZPZV<18>,
ZPZV<33>, ZPZV<55>, ZPZV<28>, ZPZV<29>, ZPZV<51>, ZPZV<6>, ZPZV<59>, ZPZV<13>, ZPZV<2>>; };  // NOLINT
04177     template<> struct ConwayPolynomial<67, 19> { using ZPZ = aerobus::zpz<67>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<65>>; };  //
NOLINT
04178     template<> struct ConwayPolynomial<71, 1> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<64>>; };  // NOLINT
04179     template<> struct ConwayPolynomial<71, 2> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7>>; };  // NOLINT
04180     template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64>>; };  // NOLINT
04181     template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, ZPZV<7>>; };  // NOLINT
04182     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
04183     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
04184     template<> struct ConwayPolynomial<71, 7> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<64>>; };  // NOLINT
04185     template<> struct ConwayPolynomial<71, 8> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<22>, ZPZV<19>, ZPZV<7>>; };  //
NOLINT
04186     template<> struct ConwayPolynomial<71, 9> { using ZPZ = aerobus::zpz<71>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43>, ZPZV<62>, ZPZV<64>>; };
```

```
    // NOLINT
04187     template<> struct ConwayPolynomial<71, 10> { using ZPZ = aerobus::zpz<71>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<17>, ZPZV<26>, ZPZV<1>, ZPZV<40>,
    ZPZV<7>>; };  // NOLINT
04188     template<> struct ConwayPolynomial<71, 11> { using ZPZ = aerobus::zpz<71>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<48>, ZPZV<64>>; };  // NOLINT
04189     template<> struct ConwayPolynomial<71, 12> { using ZPZ = aerobus::zpz<71>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<28>, ZPZV<29>, ZPZV<55>, ZPZV<21>,
    ZPZV<58>, ZPZV<23>, ZPZV<7>>; };  // NOLINT
04190     template<> struct ConwayPolynomial<71, 13> { using ZPZ = aerobus::zpz<71>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<27>, ZPZV<64>>; };  // NOLINT
04191     template<> struct ConwayPolynomial<71, 15> { using ZPZ = aerobus::zpz<71>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>,
    ZPZV<32>, ZPZV<18>, ZPZV<52>, ZPZV<67>, ZPZV<49>, ZPZV<64>>; };  // NOLINT
04192     template<> struct ConwayPolynomial<71, 17> { using ZPZ = aerobus::zpz<71>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<64>>; };  // NOLINT
04193     template<> struct ConwayPolynomial<71, 19> { using ZPZ = aerobus::zpz<71>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<64>>; };  //
    NOLINT
04194     template<> struct ConwayPolynomial<73, 1> { using ZPZ = aerobus::zpz<73>; using type =
    POLYV<ZPZV<1>, ZPZV<68>>; };  // NOLINT
04195     template<> struct ConwayPolynomial<73, 2> { using ZPZ = aerobus::zpz<73>; using type =
    POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5>>; };  // NOLINT
04196     template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68>>; };  // NOLINT
04197     template<> struct ConwayPolynomial<73, 4> { using ZPZ = aerobus::zpz<73>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5>>; };  // NOLINT
04198     template<> struct ConwayPolynomial<73, 5> { using ZPZ = aerobus::zpz<73>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<68>>; };  // NOLINT
04199     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
04200     template<> struct ConwayPolynomial<73, 7> { using ZPZ = aerobus::zpz<73>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<68>>; };  // NOLINT
04201     template<> struct ConwayPolynomial<73, 8> { using ZPZ = aerobus::zpz<73>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<53>, ZPZV<39>, ZPZV<18>, ZPZV<5>>; };  //
    NOLINT
04202     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<72>, ZPZV<15>, ZPZV<68>>; };
    // NOLINT
04203     template<> struct ConwayPolynomial<73, 10> { using ZPZ = aerobus::zpz<73>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<23>, ZPZV<33>, ZPZV<32>, ZPZV<69>,
    ZPZV<5>>; };  // NOLINT
04204     template<> struct ConwayPolynomial<73, 11> { 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<5>, ZPZV<68>>; };  // NOLINT
04205     template<> struct ConwayPolynomial<73, 12> { using ZPZ = aerobus::zpz<73>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<52>, ZPZV<26>, ZPZV<20>, ZPZV<46>,
    ZPZV<29>, ZPZV<25>, ZPZV<5>>; };  // NOLINT
04206     template<> struct ConwayPolynomial<73, 13> { 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<7>, ZPZV<68>>; };  // NOLINT
04207     template<> struct ConwayPolynomial<73, 15> { 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<10>, ZPZV<33>, ZPZV<57>, ZPZV<57>, ZPZV<62>, ZPZV<68>>; };  // NOLINT
04208     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<8>, ZPZV<68>>; };  // NOLINT
04209     template<> struct ConwayPolynomial<73, 19> { using ZPZ = aerobus::zpz<73>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<68>>; };  //
    NOLINT
04210     template<> struct ConwayPolynomial<79, 1> { using ZPZ = aerobus::zpz<79>; using type =
    POLYV<ZPZV<1>, ZPZV<76>>; };  // NOLINT
04211     template<> struct ConwayPolynomial<79, 2> { using ZPZ = aerobus::zpz<79>; using type =
    POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3>>; };  // NOLINT
04212     template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76>>; };  // NOLINT
04213     template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3>>; };  // NOLINT
04214     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
04215     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
04216     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
04217     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
04218     template<> struct ConwayPolynomial<79, 9> { using ZPZ = aerobus::zpz<79>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<19>, ZPZV<76>>; };
    // NOLINT
04219     template<> struct ConwayPolynomial<79, 10> { using ZPZ = aerobus::zpz<79>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<51>, ZPZV<1>, ZPZV<30>, ZPZV<42>,
```

```
ZPZV<3>>; };  // NOLINT
04220     template<> struct ConwayPolynomial<79, 11> { using ZPZ = aerobus::zpz<79>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<3>, ZPZV<76>>; };  // NOLINT
04221     template<> struct ConwayPolynomial<79, 12> { using ZPZ = aerobus::zpz<79>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<45>, ZPZV<52>, ZPZV<7>, ZPZV<40>,
ZPZV<59>, ZPZV<62>, ZPZV<3>>; };  // NOLINT
04222     template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<78>, ZPZV<4>, ZPZV<76>>; };  // NOLINT
04223     template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<76>>; };  // NOLINT
04224     template<> struct ConwayPolynomial<79, 19> { using ZPZ = aerobus::zpz<79>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<76>>; };  //
NOLINT
04225     template<> struct ConwayPolynomial<83, 1> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<81>>; };  // NOLINT
04226     template<> struct ConwayPolynomial<83, 2> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2>>; };  // NOLINT
04227     template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81>>; };  // NOLINT
04228     template<> struct ConwayPolynomial<83, 4> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2>>; };  // NOLINT
04229     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
04230     template<> struct ConwayPolynomial<83, 6> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<76>, ZPZV<32>, ZPZV<17>, ZPZV<2>>; };  // NOLINT
04231     template<> struct ConwayPolynomial<83, 7> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<81>>; };  // NOLINT
04232     template<> struct ConwayPolynomial<83, 8> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<23>, ZPZV<42>, ZPZV<2>>; };  //
NOLINT
04233     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
04234     template<> struct ConwayPolynomial<83, 10> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<0>, ZPZV<73>, ZPZV<0>, ZPZV<53>,
ZPZV<2>>; };  // NOLINT
04235     template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<17>, ZPZV<81>>; };  // NOLINT
04236     template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<12>, ZPZV<31>, ZPZV<19>, ZPZV<65>,
ZPZV<55>, ZPZV<75>, ZPZV<2>>; };  // NOLINT
04237     template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<81>>; };  // NOLINT
04238     template<> struct ConwayPolynomial<83, 17> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<81>>; };  // NOLINT
04239     template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<81>>; };  //
NOLINT
04240     template<> struct ConwayPolynomial<89, 1> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<86>>; };  // NOLINT
04241     template<> struct ConwayPolynomial<89, 2> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3>>; };  // NOLINT
04242     template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<86>>; };  // NOLINT
04243     template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3>>; };  // NOLINT
04244     template<> struct ConwayPolynomial<89, 5> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<86>>; };  // NOLINT
04245     template<> struct ConwayPolynomial<89, 6> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<82>, ZPZV<80>, ZPZV<15>, ZPZV<3>>; };  // NOLINT
04246     template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<86>>; };  // NOLINT
04247     template<> struct ConwayPolynomial<89, 8> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<40>, ZPZV<79>, ZPZV<3>>; };  //
NOLINT
04248     template<> struct ConwayPolynomial<89, 9> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<12>, ZPZV<6>, ZPZV<86>>; };
// NOLINT
04249     template<> struct ConwayPolynomial<89, 10> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<33>, ZPZV<82>, ZPZV<52>, ZPZV<4>,
ZPZV<3>>; };  // NOLINT
04250     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<88>,
ZPZV<26>, ZPZV<86>>; };  // NOLINT
04251     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
04252     template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
```

```
ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<86>>; };  // NOLINT
04253     template<> struct ConwayPolynomial<89, 17> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<86>>; };  // NOLINT
04254     template<> struct ConwayPolynomial<89, 19> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<86>>; };  //
NOLINT
04255     template<> struct ConwayPolynomial<97, 1> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<92>>; };  // NOLINT
04256     template<> struct ConwayPolynomial<97, 2> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5>>; };  // NOLINT
04257     template<> struct ConwayPolynomial<97, 3> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<92>>; };  // NOLINT
04258     template<> struct ConwayPolynomial<97, 4> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5>>; };  // NOLINT
04259     template<> struct ConwayPolynomial<97, 5> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<92>>; };  // NOLINT
04260     template<> struct ConwayPolynomial<97, 6> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<92>, ZPZV<58>, ZPZV<88>, ZPZV<5>>; };  // NOLINT
04261     template<> struct ConwayPolynomial<97, 7> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<92>>; };  // NOLINT
04262     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>>; };  //
NOLINT
04263     template<> struct ConwayPolynomial<97, 9> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59>, ZPZV<7>, ZPZV<92>>; };
// NOLINT
04264     template<> struct ConwayPolynomial<97, 10> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<66>, ZPZV<34>, ZPZV<34>, ZPZV<20>,
ZPZV<5>>; };  // NOLINT
04265     template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<5>, ZPZV<92>>; };  // NOLINT
04266     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<0>, ZPZV<86>,
ZPZV<78>, ZPZV<94>, ZPZV<5>>; };  // NOLINT
04267     template<> struct ConwayPolynomial<97, 13> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<92>>; };  // NOLINT
04268     template<> struct ConwayPolynomial<97, 17> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<92>>; };  // NOLINT
04269     template<> struct ConwayPolynomial<97, 19> { using ZPZ = aerobus::zpz<97>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<92>>; };  //
NOLINT
04270     template<> struct ConwayPolynomial<101, 1> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<99>>; };  // NOLINT
04271     template<> struct ConwayPolynomial<101, 2> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2>>; };  // NOLINT
04272     template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<99>>; };  // NOLINT
04273     template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<78>, ZPZV<2>>; };  // NOLINT
04274     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
04275     template<> struct ConwayPolynomial<101, 6> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<90>, ZPZV<20>, ZPZV<67>, ZPZV<2>>; };  // NOLINT
04276     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<99>>; };  // NOLINT
04277     template<> struct ConwayPolynomial<101, 8> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76>, ZPZV<29>, ZPZV<24>, ZPZV<2>>; };  //
NOLINT
04278     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<64>, ZPZV<47>, ZPZV<99>>; };
// NOLINT
04279     template<> struct ConwayPolynomial<101, 10> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<67>, ZPZV<49>, ZPZV<100>, ZPZV<100>, ZPZV<52>,
ZPZV<2>>; };  // NOLINT
04280     template<> struct ConwayPolynomial<101, 11> { 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<31>, ZPZV<99>>; };  // NOLINT
04281     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<78>, ZPZV<48>,
ZPZV<84>, ZPZV<21>, ZPZV<2>>; };  // NOLINT
04282     template<> struct ConwayPolynomial<101, 13> { using ZPZ = aerobus::zpz<101>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<99>>; };  // NOLINT
04283     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<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<99>>; };  // NOLINT
04284     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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<99>>; };  //
NOLINT
04285     template<> struct ConwayPolynomial<103, 1> { using ZPZ = aerobus::zpz<103>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<98>>; };  // NOLINT
04286     template<> struct ConwayPolynomial<103, 2> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5>>; };  // NOLINT
04287     template<> struct ConwayPolynomial<103, 3> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<98>>; };  // NOLINT
04288     template<> struct ConwayPolynomial<103, 4> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<88>, ZPZV<5>>; };  // NOLINT
04289     template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<98>>; };  // NOLINT
04290     template<> struct ConwayPolynomial<103, 6> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<9>, ZPZV<30>, ZPZV<5>>; };  // NOLINT
04291     template<> struct ConwayPolynomial<103, 7> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98>>; };  // NOLINT
04292     template<> struct ConwayPolynomial<103, 8> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<70>, ZPZV<71>, ZPZV<49>, ZPZV<5>>; };  //
      NOLINT
04293     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<97>, ZPZV<51>, ZPZV<98>>; };
      // NOLINT
04294     template<> struct ConwayPolynomial<103, 10> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<101>, ZPZV<86>, ZPZV<101>, ZPZV<94>, ZPZV<11>,
      ZPZV<5>>; };  // NOLINT
04295     template<> struct ConwayPolynomial<103, 11> { 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<5>, ZPZV<98>>; };  // NOLINT
04296     template<> struct ConwayPolynomial<103, 12> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<74>, ZPZV<23>, ZPZV<94>, ZPZV<20>, ZPZV<81>,
      ZPZV<29>, ZPZV<88>, ZPZV<5>>; };  // NOLINT
04297     template<> struct ConwayPolynomial<103, 13> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98>>; };  // NOLINT
04298     template<> struct ConwayPolynomial<103, 17> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<102>, ZPZV<8>, ZPZV<98>>; };  // NOLINT
04299     template<> struct ConwayPolynomial<103, 19> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<98>>; };  //
      NOLINT
04300     template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<105>>; };  // NOLINT
04301     template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<103>, ZPZV<2>>; };  // NOLINT
04302     template<> struct ConwayPolynomial<107, 3> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<105>>; };  // NOLINT
04303     template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2>>; };  // NOLINT
04304     template<> struct ConwayPolynomial<107, 5> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<105>>; };  // NOLINT
04305     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
04306     template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<105>>; };  // NOLINT
04307     template<> struct ConwayPolynomial<107, 8> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<24>, ZPZV<95>, ZPZV<2>>; };  //
      NOLINT
04308     template<> struct ConwayPolynomial<107, 9> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<66>, ZPZV<105>>; };
      // NOLINT
04309     template<> struct ConwayPolynomial<107, 10> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<94>, ZPZV<61>, ZPZV<83>, ZPZV<83>, ZPZV<95>,
      ZPZV<2>>; };  // NOLINT
04310     template<> struct ConwayPolynomial<107, 11> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<8>, ZPZV<105>>; };  // NOLINT
04311     template<> struct ConwayPolynomial<107, 12> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<37>, ZPZV<48>, ZPZV<6>, ZPZV<0>, ZPZV<61>,
      ZPZV<42>, ZPZV<57>, ZPZV<2>>; };  // NOLINT
04312     template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105>>; };  // NOLINT
04313     template<> struct ConwayPolynomial<107, 17> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>>; };  // NOLINT
04314     template<> struct ConwayPolynomial<107, 19> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<105>>; };  //
      NOLINT
04315     template<> struct ConwayPolynomial<109, 1> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<103>>; };  // NOLINT
04316     template<> struct ConwayPolynomial<109, 2> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6>>; };  // NOLINT
04317     template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103>>; };  // NOLINT
04318     template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6>>; };  // NOLINT
04319     template<> struct ConwayPolynomial<109, 5> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>>; };  // NOLINT
```

```
04320     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
04321     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<103>>; };  // NOLINT
04322     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
04323     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<93>, ZPZV<87>, ZPZV<103>>;
      };  // NOLINT
04324     template<> struct ConwayPolynomial<109, 10> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<71>, ZPZV<55>, ZPZV<16>, ZPZV<75>, ZPZV<69>,
      ZPZV<6>>; };  // NOLINT
04325     template<> struct ConwayPolynomial<109, 11> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<11>, ZPZV<103>>; };  // NOLINT
04326     template<> struct ConwayPolynomial<109, 12> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<50>, ZPZV<53>, ZPZV<37>, ZPZV<8>, ZPZV<65>,
      ZPZV<103>, ZPZV<28>, ZPZV<6>>; };  // NOLINT
04327     template<> struct ConwayPolynomial<109, 13> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<103>>; };  // NOLINT
04328     template<> struct ConwayPolynomial<109, 17> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<103>>; };  // NOLINT
04329     template<> struct ConwayPolynomial<109, 19> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<103>>; };  //
      NOLINT
04330     template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<110>>; };  // NOLINT
04331     template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3>>; };  // NOLINT
04332     template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<110>>; };  // NOLINT
04333     template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<3>>; };  // NOLINT
04334     template<> struct ConwayPolynomial<113, 5> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<110>>; };  // NOLINT
04335     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
04336     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<110>>; };  // NOLINT
04337     template<> struct ConwayPolynomial<113, 8> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<98>, ZPZV<38>, ZPZV<28>, ZPZV<3>>; };  //
      NOLINT
04338     template<> struct ConwayPolynomial<113, 9> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<87>, ZPZV<71>, ZPZV<110>>;
      };  // NOLINT
04339     template<> struct ConwayPolynomial<113, 10> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<108>, ZPZV<57>, ZPZV<45>, ZPZV<83>, ZPZV<56>,
      ZPZV<3>>; };  // NOLINT
04340     template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<3>, ZPZV<110>>; };  // NOLINT
04341     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
04342     template<> struct ConwayPolynomial<113, 13> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110>>; };  // NOLINT
04343     template<> struct ConwayPolynomial<113, 17> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110>>; };  // NOLINT
04344     template<> struct ConwayPolynomial<113, 19> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<110>>; };  //
      NOLINT
04345     template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<124>>; };  // NOLINT
04346     template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3>>; };  // NOLINT
04347     template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124>>; };  // NOLINT
04348     template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<97>, ZPZV<3>>; };  // NOLINT
04349     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
04350     template<> struct ConwayPolynomial<127, 6> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<84>, ZPZV<115>, ZPZV<82>, ZPZV<3>>; };  // NOLINT
04351     template<> struct ConwayPolynomial<127, 7> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<124>>; };  // NOLINT
04352     template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<104>, ZPZV<55>, ZPZV<8>, ZPZV<3>>; };  //
      NOLINT
04353     template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<119>, ZPZV<126>,
```

```
ZPZV<124>>; };  // NOLINT
04354     template<> struct ConwayPolynomial<127, 10> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<107>, ZPZV<64>, ZPZV<95>, ZPZV<60>, ZPZV<4>,
ZPZV<3>>; };  // NOLINT
04355     template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<11>, ZPZV<124>>; };  // NOLINT
04356     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
04357     template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<124>>; };  // NOLINT
04358     template<> struct ConwayPolynomial<127, 17> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<124>>; };  // NOLINT
04359     template<> struct ConwayPolynomial<127, 19> { using ZPZ = aerobus::zpz<127>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<124>>; };  //
NOLINT
04360     template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<129>>; };  // NOLINT
04361     template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2>>; };  // NOLINT
04362     template<> struct ConwayPolynomial<131, 3> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<129>>; };  // NOLINT
04363     template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2>>; };  // NOLINT
04364     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
04365     template<> struct ConwayPolynomial<131, 6> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<4>, ZPZV<22>, ZPZV<2>>; };  // NOLINT
04366     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<10>, ZPZV<129>>; };  // NOLINT
04367     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
04368     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<19>, ZPZV<129>>; };
// NOLINT
04369     template<> struct ConwayPolynomial<131, 10> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<124>, ZPZV<97>, ZPZV<9>, ZPZV<126>, ZPZV<44>,
ZPZV<2>>; };  // NOLINT
04370     template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<6>, ZPZV<129>>; };  // NOLINT
04371     template<> struct ConwayPolynomial<131, 12> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<122>, ZPZV<40>, ZPZV<83>, ZPZV<125>,
ZPZV<28>, ZPZV<103>, ZPZV<2>>; };  // NOLINT
04372     template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<129>>; };  // NOLINT
04373     template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<129>>; };  // NOLINT
04374     template<> struct ConwayPolynomial<131, 19> { using ZPZ = aerobus::zpz<131>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<129>>; };  //
NOLINT
04375     template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<134>>; };  // NOLINT
04376     template<> struct ConwayPolynomial<137, 2> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3>>; };  // NOLINT
04377     template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134>>; };  // NOLINT
04378     template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<95>, ZPZV<3>>; };  // NOLINT
04379     template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<134>>; };  // NOLINT
04380     template<> struct ConwayPolynomial<137, 6> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<116>, ZPZV<102>, ZPZV<3>, ZPZV<3>>; };  // NOLINT
04381     template<> struct ConwayPolynomial<137, 7> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<134>>; };  // NOLINT
04382     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<3>>; };  //
NOLINT
04383     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
04384     template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<20>, ZPZV<67>, ZPZV<93>, ZPZV<119>,
ZPZV<3>>; };  // NOLINT
04385     template<> struct ConwayPolynomial<137, 11> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<1>, ZPZV<134>>; };  // NOLINT
04386     template<> struct ConwayPolynomial<137, 12> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<61>, ZPZV<40>, ZPZV<40>, ZPZV<12>, ZPZV<36>,
```

```
ZPZV<135>, ZPZV<61>, ZPZV<3>>; };  // NOLINT
04387     template<> struct ConwayPolynomial<137, 13> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<134>>; };  // NOLINT
04388     template<> struct ConwayPolynomial<137, 17> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<4>, ZPZV<134>>; };  // NOLINT
04389     template<> struct ConwayPolynomial<137, 19> { using ZPZ = aerobus::zpz<137>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<134>>; };  //
NOLINT
04390     template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<137>>; };  // NOLINT
04391     template<> struct ConwayPolynomial<139, 2> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2>>; };  // NOLINT
04392     template<> struct ConwayPolynomial<139, 3> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<137>>; };  // NOLINT
04393     template<> struct ConwayPolynomial<139, 4> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2>>; };  // NOLINT
04394     template<> struct ConwayPolynomial<139, 5> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<137>>; };  // NOLINT
04395     template<> struct ConwayPolynomial<139, 6> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<46>, ZPZV<10>, ZPZV<118>, ZPZV<2>>; };  // NOLINT
04396     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<9>, ZPZV<137>>; };  // NOLINT
04397     template<> struct ConwayPolynomial<139, 8> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>, ZPZV<36>, ZPZV<21>, ZPZV<2>>; };  //
NOLINT
04398     template<> struct ConwayPolynomial<139, 9> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<70>, ZPZV<87>, ZPZV<137>>;
};  // NOLINT
04399     template<> struct ConwayPolynomial<139, 10> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<110>, ZPZV<48>, ZPZV<130>, ZPZV<66>,
ZPZV<106>, ZPZV<2>>; };  // NOLINT
04400     template<> struct ConwayPolynomial<139, 11> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<7>, ZPZV<137>>; };  // NOLINT
04401     template<> struct ConwayPolynomial<139, 12> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<120>, ZPZV<75>, ZPZV<41>, ZPZV<77>, ZPZV<106>,
ZPZV<8>, ZPZV<10>, ZPZV<2>>; };  // NOLINT
04402     template<> struct ConwayPolynomial<139, 13> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<137>>; };  // NOLINT
04403     template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137>>; };  // NOLINT
04404     template<> struct ConwayPolynomial<139, 19> { using ZPZ = aerobus::zpz<139>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<137>>; };  //
NOLINT
04405     template<> struct ConwayPolynomial<149, 1> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<147>>; };  // NOLINT
04406     template<> struct ConwayPolynomial<149, 2> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<145>, ZPZV<2>>; };  // NOLINT
04407     template<> struct ConwayPolynomial<149, 3> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147>>; };  // NOLINT
04408     template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<2>>; };  // NOLINT
04409     template<> struct ConwayPolynomial<149, 5> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<147>>; };  // NOLINT
04410     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<2>>; };  // NOLINT
04411     template<> struct ConwayPolynomial<149, 7> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<147>>; };  // NOLINT
04412     template<> struct ConwayPolynomial<149, 8> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<140>, ZPZV<25>, ZPZV<123>, ZPZV<2>>; };  //
NOLINT
04413     template<> struct ConwayPolynomial<149, 9> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<146>, ZPZV<20>, ZPZV<147>>;
};  // NOLINT
04414     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
04415     template<> struct ConwayPolynomial<149, 11> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<33>, ZPZV<147>>; };  // NOLINT
04416     template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<91>, ZPZV<52>, ZPZV<9>,
ZPZV<104>, ZPZV<110>, ZPZV<2>>; };  // NOLINT
04417     template<> struct ConwayPolynomial<149, 13> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<147>>; };  // NOLINT
04418     template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<147>>; };  // NOLINT
04419     template<> struct ConwayPolynomial<149, 19> { using ZPZ = aerobus::zpz<149>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
```

```
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<147>>; };  //
      NOLINT
04420     template<> struct ConwayPolynomial<151, 1> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<145>>; };  // NOLINT
04421     template<> struct ConwayPolynomial<151, 2> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6>>; };  // NOLINT
04422     template<> struct ConwayPolynomial<151, 3> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<145>>; };  // NOLINT
04423     template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6>>; };  // NOLINT
04424     template<> struct ConwayPolynomial<151, 5> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<145>>; };  // NOLINT
04425     template<> struct ConwayPolynomial<151, 6> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<125>, ZPZV<18>, ZPZV<15>, ZPZV<6>>; };  // NOLINT
04426     template<> struct ConwayPolynomial<151, 7> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<145>>; };  // NOLINT
04427     template<> struct ConwayPolynomial<151, 8> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<140>, ZPZV<122>, ZPZV<43>, ZPZV<6>>; };  //
      NOLINT
04428     template<> struct ConwayPolynomial<151, 9> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<126>, ZPZV<96>, ZPZV<145>>;
      };  // NOLINT
04429     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
04430     template<> struct ConwayPolynomial<151, 11> { 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<1>, ZPZV<145>>; };  // NOLINT
04431     template<> struct ConwayPolynomial<151, 12> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<121>, ZPZV<101>, ZPZV<6>, ZPZV<77>,
      ZPZV<107>, ZPZV<147>, ZPZV<6>>; };  // NOLINT
04432     template<> struct ConwayPolynomial<151, 13> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<145>>; };  // NOLINT
04433     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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<145>>; };  // NOLINT
04434     template<> struct ConwayPolynomial<151, 19> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<145>>; };  //
      NOLINT
04435     template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<152>>; };  // NOLINT
04436     template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5>>; };  // NOLINT
04437     template<> struct ConwayPolynomial<157, 3> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<152>>; };  // NOLINT
04438     template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<136>, ZPZV<5>>; };  // NOLINT
04439     template<> struct ConwayPolynomial<157, 5> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<152>>; };  // NOLINT
04440     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
04441     template<> struct ConwayPolynomial<157, 7> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152>>; };  // NOLINT
04442     template<> struct ConwayPolynomial<157, 8> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<97>, ZPZV<40>, ZPZV<153>, ZPZV<5>>; };  //
      NOLINT
04443     template<> struct ConwayPolynomial<157, 9> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<114>, ZPZV<52>, ZPZV<152>>;
      };  // NOLINT
04444     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>,
      ZPZV<5>>; };  // NOLINT
04445     template<> struct ConwayPolynomial<157, 11> { 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<29>, ZPZV<152>>; };  // NOLINT
04446     template<> struct ConwayPolynomial<157, 12> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<77>, ZPZV<110>, ZPZV<72>, ZPZV<137>, ZPZV<43>,
      ZPZV<152>, ZPZV<57>, ZPZV<5>>; };  // NOLINT
04447     template<> struct ConwayPolynomial<157, 13> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<156>, ZPZV<9>, ZPZV<152>>; };  // NOLINT
04448     template<> struct ConwayPolynomial<157, 17> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<152>>; };  // NOLINT
04449     template<> struct ConwayPolynomial<157, 19> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152>>; };  //
      NOLINT
04450     template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
      POLYV<ZPZV<1>, ZPZV<161>>; };  // NOLINT
04451     template<> struct ConwayPolynomial<163, 2> { using ZPZ = aerobus::zpz<163>; using type =
      POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2>>; };  // NOLINT
04452     template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<161>>; };  // NOLINT
04453     template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
```

```
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2>>; };  // NOLINT
04454     template<> struct ConwayPolynomial<163, 5> { using ZPZ = aerobus::zpz<163>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<161>>; };  // NOLINT
04455     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
04456     template<> struct ConwayPolynomial<163, 7> { using ZPZ = aerobus::zpz<163>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<161>>; };  // NOLINT
04457     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>>; };  //
    NOLINT
04458     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<15>, ZPZV<162>, ZPZV<127>,
    ZPZV<161>>; };  // NOLINT
04459     template<> struct ConwayPolynomial<163, 10> { using ZPZ = aerobus::zpz<163>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<111>, ZPZV<120>, ZPZV<125>, ZPZV<15>, ZPZV<0>,
    ZPZV<2>>; };  // NOLINT
04460     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<11>, ZPZV<161>>; };  // NOLINT
04461     template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<112>, ZPZV<31>, ZPZV<38>, ZPZV<103>,
    ZPZV<10>, ZPZV<69>, ZPZV<2>>; };  // NOLINT
04462     template<> struct ConwayPolynomial<163, 13> { 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<24>, ZPZV<161>>; };  // NOLINT
04463     template<> struct ConwayPolynomial<163, 17> { using ZPZ = aerobus::zpz<163>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<71>, ZPZV<161>>; };  // NOLINT
04464     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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<161>>; };  //
    NOLINT
04465     template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<162>>; };  // NOLINT
04466     template<> struct ConwayPolynomial<167, 2> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<166>, ZPZV<5>>; };  // NOLINT
04467     template<> struct ConwayPolynomial<167, 3> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>>; };  // NOLINT
04468     template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5>>; };  // NOLINT
04469     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
04470     template<> struct ConwayPolynomial<167, 6> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<75>, ZPZV<38>, ZPZV<2>, ZPZV<5>>; };  // NOLINT
04471     template<> struct ConwayPolynomial<167, 7> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162>>; };  // NOLINT
04472     template<> struct ConwayPolynomial<167, 8> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<149>, ZPZV<56>, ZPZV<113>, ZPZV<5>>; };  //
    NOLINT
04473     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<165>, ZPZV<122>, ZPZV<162>>;
    };  // NOLINT
04474     template<> struct ConwayPolynomial<167, 10> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<85>, ZPZV<68>, ZPZV<109>, ZPZV<143>,
    ZPZV<148>, ZPZV<5>>; };  // NOLINT
04475     template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<24>, ZPZV<162>>; };  // NOLINT
04476     template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<142>, ZPZV<10>, ZPZV<142>, ZPZV<131>,
    ZPZV<140>, ZPZV<41>, ZPZV<57>, ZPZV<5>>; };  // NOLINT
04477     template<> struct ConwayPolynomial<167, 13> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162>>; };  // NOLINT
04478     template<> struct ConwayPolynomial<167, 17> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<162>>; };  // NOLINT
04479     template<> struct ConwayPolynomial<167, 19> { using ZPZ = aerobus::zpz<167>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<162>>; };  //
    NOLINT
04480     template<> struct ConwayPolynomial<173, 1> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<171>>; };  // NOLINT
04481     template<> struct ConwayPolynomial<173, 2> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<169>, ZPZV<2>>; };  // NOLINT
04482     template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171>>; };  // NOLINT
04483     template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<2>>; };  // NOLINT
04484     template<> struct ConwayPolynomial<173, 5> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171>>; };  // NOLINT
04485     template<> struct ConwayPolynomial<173, 6> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<134>, ZPZV<107>, ZPZV<2>>; };  // NOLINT
04486     template<> struct ConwayPolynomial<173, 7> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<171>>; };  // NOLINT
04487     template<> struct ConwayPolynomial<173, 8> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<125>, ZPZV<158>, ZPZV<27>, ZPZV<2>>; };  //
```

```
    NOLINT
04488     template<> struct ConwayPolynomial<173, 9> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<104>, ZPZV<171>>;
    };  // NOLINT
04489     template<> struct ConwayPolynomial<173, 10> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<156>, ZPZV<164>, ZPZV<48>, ZPZV<106>,
    ZPZV<58>, ZPZV<2>>; };  // NOLINT
04490     template<> struct ConwayPolynomial<173, 11> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<12>, ZPZV<171>>; };  // NOLINT
04491     template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<64>, ZPZV<46>, ZPZV<166>, ZPZV<0>,
    ZPZV<159>, ZPZV<22>, ZPZV<2>>; };  // NOLINT
04492     template<> struct ConwayPolynomial<173, 13> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171>>; };  // NOLINT
04493     template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<171>>; };  // NOLINT
04494     template<> struct ConwayPolynomial<173, 19> { using ZPZ = aerobus::zpz<173>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171>>; };  //
    NOLINT
04495     template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<177>>; };  // NOLINT
04496     template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2>>; };  // NOLINT
04497     template<> struct ConwayPolynomial<179, 3> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<177>>; };  // NOLINT
04498     template<> struct ConwayPolynomial<179, 4> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<2>>; };  // NOLINT
04499     template<> struct ConwayPolynomial<179, 5> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<177>>; };  // NOLINT
04500     template<> struct ConwayPolynomial<179, 6> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<91>, ZPZV<55>, ZPZV<109>, ZPZV<2>>; };  // NOLINT
04501     template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<177>>; };  // NOLINT
04502     template<> struct ConwayPolynomial<179, 8> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<163>, ZPZV<144>, ZPZV<73>, ZPZV<2>>; };  //
    NOLINT
04503     template<> struct ConwayPolynomial<179, 9> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<64>, ZPZV<177>>;
    };  // NOLINT
04504     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
04505     template<> struct ConwayPolynomial<179, 11> { 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<28>, ZPZV<177>>; };  // NOLINT
04506     template<> struct ConwayPolynomial<179, 12> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<103>, ZPZV<83>, ZPZV<43>, ZPZV<76>, ZPZV<8>,
    ZPZV<177>, ZPZV<1>, ZPZV<2>>; };  // NOLINT
04507     template<> struct ConwayPolynomial<179, 13> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<177>>; };  // NOLINT
04508     template<> struct ConwayPolynomial<179, 17> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177>>; };  // NOLINT
04509     template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<177>>; };  //
    NOLINT
04510     template<> struct ConwayPolynomial<181, 1> { using ZPZ = aerobus::zpz<181>; using type =
    POLYV<ZPZV<1>, ZPZV<179>>; };  // NOLINT
04511     template<> struct ConwayPolynomial<181, 2> { using ZPZ = aerobus::zpz<181>; using type =
    POLYV<ZPZV<1>, ZPZV<177>, ZPZV<2>>; };  // NOLINT
04512     template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179>>; };  // NOLINT
04513     template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<105>, ZPZV<2>>; };  // NOLINT
04514     template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<179>>; };  // NOLINT
04515     template<> struct ConwayPolynomial<181, 6> { using ZPZ = aerobus::zpz<181>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<163>, ZPZV<169>, ZPZV<2>>; };  // NOLINT
04516     template<> struct ConwayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<179>>; };  // NOLINT
04517     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
04518     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>>; };  // NOLINT
04519     template<> struct ConwayPolynomial<181, 10> { using ZPZ = aerobus::zpz<181>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<154>, ZPZV<104>, ZPZV<94>, ZPZV<57>, ZPZV<88>,
    ZPZV<2>>; };  // NOLINT
04520     template<> struct ConwayPolynomial<181, 11> { 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<24>, ZPZV<179>>; };  // NOLINT
04521     template<> struct ConwayPolynomial<181, 12> { using ZPZ = aerobus::zpz<181>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<141>, ZPZV<45>, ZPZV<122>,
ZPZV<175>, ZPZV<12>, ZPZV<10>, ZPZV<2>>; };  // NOLINT
04522     template<> struct ConwayPolynomial<181, 13> { 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<8>, ZPZV<179>>; };  // NOLINT
04523     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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<179>>; };  // NOLINT
04524     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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<179>>; };  //
NOLINT
04525     template<> struct ConwayPolynomial<191, 1> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<172>>; };  // NOLINT
04526     template<> struct ConwayPolynomial<191, 2> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19>>; };  // NOLINT
04527     template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172>>; };  // NOLINT
04528     template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19>>; };  // NOLINT
04529     template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<172>>; };  // NOLINT
04530     template<> struct ConwayPolynomial<191, 6> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<110>, ZPZV<10>, ZPZV<10>, ZPZV<19>>; };  // NOLINT
04531     template<> struct ConwayPolynomial<191, 7> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<172>>; };  // NOLINT
04532     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
04533     template<> struct ConwayPolynomial<191, 9> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<124>, ZPZV<172>>;
};  // NOLINT
04534     template<> struct ConwayPolynomial<191, 10> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<113>, ZPZV<47>, ZPZV<173>, ZPZV<74>,
ZPZV<156>, ZPZV<19>>; };  // NOLINT
04535     template<> struct ConwayPolynomial<191, 11> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<6>, ZPZV<172>>; };  // NOLINT
04536     template<> struct ConwayPolynomial<191, 12> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<79>, ZPZV<168>, ZPZV<25>, ZPZV<49>, ZPZV<90>,
ZPZV<7>, ZPZV<151>, ZPZV<19>>; };  // NOLINT
04537     template<> struct ConwayPolynomial<191, 13> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<172>>; };  // NOLINT
04538     template<> struct ConwayPolynomial<191, 17> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<172>>; };  // NOLINT
04539     template<> struct ConwayPolynomial<191, 19> { using ZPZ = aerobus::zpz<191>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<190>, ZPZV<2>, ZPZV<172>>; };  //
NOLINT
04540     template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<188>>; };  // NOLINT
04541     template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5>>; };  // NOLINT
04542     template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188>>; };  // NOLINT
04543     template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<5>>; };  // NOLINT
04544     template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<188>>; };  // NOLINT
04545     template<> struct ConwayPolynomial<193, 6> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<8>, ZPZV<172>, ZPZV<5>>; };  // NOLINT
04546     template<> struct ConwayPolynomial<193, 7> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<188>>; };  // NOLINT
04547     template<> struct ConwayPolynomial<193, 8> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<145>, ZPZV<34>, ZPZV<154>, ZPZV<5>>; };  //
NOLINT
04548     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
04549     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
04550     template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<1>, ZPZV<188>>; };  // NOLINT
04551     template<> struct ConwayPolynomial<193, 12> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<155>, ZPZV<52>, ZPZV<135>, ZPZV<152>,
ZPZV<90>, ZPZV<46>, ZPZV<28>, ZPZV<5>>; };  // NOLINT
04552     template<> struct ConwayPolynomial<193, 13> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<188>>; };  // NOLINT
04553     template<> struct ConwayPolynomial<193, 17> { using ZPZ = aerobus::zpz<193>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
```

```
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<188>>; };  // NOLINT
04554     template<> struct ConwayPolynomial<193, 19> { using ZPZ = aerobus::zpz<193>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<188>>; };  //
    NOLINT
04555     template<> struct ConwayPolynomial<197, 1> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<195>>; };  // NOLINT
04556     template<> struct ConwayPolynomial<197, 2> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2>>; };  // NOLINT
04557     template<> struct ConwayPolynomial<197, 3> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<195>>; };  // NOLINT
04558     template<> struct ConwayPolynomial<197, 4> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<124>, ZPZV<2>>; };  // NOLINT
04559     template<> struct ConwayPolynomial<197, 5> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195>>; };  // NOLINT
04560     template<> struct ConwayPolynomial<197, 6> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<124>, ZPZV<79>, ZPZV<173>, ZPZV<2>>; };  // NOLINT
04561     template<> struct ConwayPolynomial<197, 7> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<195>>; };  // NOLINT
04562     template<> struct ConwayPolynomial<197, 8> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<176>, ZPZV<96>, ZPZV<29>, ZPZV<2>>; };  //
    NOLINT
04563     template<> struct ConwayPolynomial<197, 9> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<127>, ZPZV<8>, ZPZV<195>>;
    };  // NOLINT
04564     template<> struct ConwayPolynomial<197, 10> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<137>, ZPZV<8>, ZPZV<73>, ZPZV<42>,
    ZPZV<2>>; };  // NOLINT
04565     template<> struct ConwayPolynomial<197, 11> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<14>, ZPZV<195>>; };  // NOLINT
04566     template<> struct ConwayPolynomial<197, 12> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<168>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>,
    ZPZV<90>, ZPZV<163>, ZPZV<2>>; };  // NOLINT
04567     template<> struct ConwayPolynomial<197, 13> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<195>>; };  // NOLINT
04568     template<> struct ConwayPolynomial<197, 17> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<195>>; };  // NOLINT
04569     template<> struct ConwayPolynomial<197, 19> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<195>>; };  //
    NOLINT
04570     template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<196>>; };  // NOLINT
04571     template<> struct ConwayPolynomial<199, 2> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<193>, ZPZV<3>>; };  // NOLINT
04572     template<> struct ConwayPolynomial<199, 3> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196>>; };  // NOLINT
04573     template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3>>; };  // NOLINT
04574     template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196>>; };  // NOLINT
04575     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
04576     template<> struct ConwayPolynomial<199, 7> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196>>; };  // NOLINT
04577     template<> struct ConwayPolynomial<199, 8> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<160>, ZPZV<23>, ZPZV<159>, ZPZV<3>>; };  //
    NOLINT
04578     template<> struct ConwayPolynomial<199, 9> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<177>, ZPZV<141>, ZPZV<196>>;
    };  // NOLINT
04579     template<> struct ConwayPolynomial<199, 10> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<158>, ZPZV<31>, ZPZV<54>, ZPZV<9>,
    ZPZV<3>>; };  // NOLINT
04580     template<> struct ConwayPolynomial<199, 11> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<1>, ZPZV<196>>; };  // NOLINT
04581     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
04582     template<> struct ConwayPolynomial<199, 13> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<196>>; };  // NOLINT
04583     template<> struct ConwayPolynomial<199, 17> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<196>>; };  // NOLINT
04584     template<> struct ConwayPolynomial<199, 19> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<196>>; };  //
    NOLINT
04585     template<> struct ConwayPolynomial<211, 1> { using ZPZ = aerobus::zpz<211>; using type =
    POLYV<ZPZV<1>, ZPZV<209>>; };  // NOLINT
04586     template<> struct ConwayPolynomial<211, 2> { using ZPZ = aerobus::zpz<211>; using type =
    POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2>>; };  // NOLINT
```

```
04587     template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<209>>; };  // NOLINT
04588     template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2>>; };  // NOLINT
04589     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
04590     template<> struct ConwayPolynomial<211, 6> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<81>, ZPZV<194>, ZPZV<133>, ZPZV<2>>; };  // NOLINT
04591     template<> struct ConwayPolynomial<211, 7> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<209>>; };  // NOLINT
04592     template<> struct ConwayPolynomial<211, 8> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<200>, ZPZV<87>, ZPZV<29>, ZPZV<2>>; };  //
      NOLINT
04593     template<> struct ConwayPolynomial<211, 9> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<139>, ZPZV<26>, ZPZV<209>>;
      };  // NOLINT
04594     template<> struct ConwayPolynomial<211, 10> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<30>, ZPZV<61>, ZPZV<148>, ZPZV<87>, ZPZV<125>,
      ZPZV<2>>; };  // NOLINT
04595     template<> struct ConwayPolynomial<211, 11> { 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<7>, ZPZV<209>>; };  // NOLINT
04596     template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<50>, ZPZV<145>, ZPZV<126>, ZPZV<184>,
      ZPZV<84>, ZPZV<27>, ZPZV<2>>; };  // NOLINT
04597     template<> struct ConwayPolynomial<211, 13> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<12>, ZPZV<209>>; };  // NOLINT
04598     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<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<209>>; };  // NOLINT
04599     template<> struct ConwayPolynomial<211, 19> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<209>>; };  //
      NOLINT
04600     template<> struct ConwayPolynomial<223, 1> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<220>>; };  // NOLINT
04601     template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3>>; };  // NOLINT
04602     template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<220>>; };  // NOLINT
04603     template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<3>>; };  // NOLINT
04604     template<> struct ConwayPolynomial<223, 5> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220>>; };  // NOLINT
04605     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
04606     template<> struct ConwayPolynomial<223, 7> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<220>>; };  // NOLINT
04607     template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<3>>; };  //
      NOLINT
04608     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
04609     template<> struct ConwayPolynomial<223, 10> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<118>, ZPZV<177>, ZPZV<87>, ZPZV<99>, ZPZV<62>,
      ZPZV<3>>; };  // NOLINT
04610     template<> struct ConwayPolynomial<223, 11> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<8>, ZPZV<220>>; };  // NOLINT
04611     template<> struct ConwayPolynomial<223, 12> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<64>, ZPZV<94>, ZPZV<11>, ZPZV<105>, ZPZV<64>,
      ZPZV<151>, ZPZV<213>, ZPZV<3>>; };  // NOLINT
04612     template<> struct ConwayPolynomial<223, 13> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<220>>; };  // NOLINT
04613     template<> struct ConwayPolynomial<223, 17> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220>>; };  // NOLINT
04614     template<> struct ConwayPolynomial<223, 19> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<220>>; };  //
      NOLINT
04615     template<> struct ConwayPolynomial<227, 1> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<225>>; };  // NOLINT
04616     template<> struct ConwayPolynomial<227, 2> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2>>; };  // NOLINT
04617     template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<225>>; };  // NOLINT
04618     template<> struct ConwayPolynomial<227, 4> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2>>; };  // NOLINT
04619     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
04620     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
04621     template<> struct ConwayPolynomial<227, 7> { using ZPZ = aerobus::zpz<227>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<225>>; };  // NOLINT
04622     template<> struct ConwayPolynomial<227, 8> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<151>, ZPZV<176>, ZPZV<106>, ZPZV<2>>; };  //
      NOLINT
04623     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<3>, ZPZV<24>, ZPZV<183>, ZPZV<225>>;
      };  // NOLINT
04624     template<> struct ConwayPolynomial<227, 10> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<199>, ZPZV<12>, ZPZV<93>, ZPZV<77>,
      ZPZV<2>>; };  // NOLINT
04625     template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<2>, ZPZV<225>>; };  // NOLINT
04626     template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<123>, ZPZV<99>, ZPZV<160>, ZPZV<96>,
      ZPZV<127>, ZPZV<142>, ZPZV<94>, ZPZV<2>>; };  // NOLINT
04627     template<> struct ConwayPolynomial<227, 13> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<225>>; };  // NOLINT
04628     template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<225>>; };  // NOLINT
04629     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>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<225>>; };  //
      NOLINT
04630     template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<223>>; };  // NOLINT
04631     template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6>>; };  // NOLINT
04632     template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223>>; };  // NOLINT
04633     template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<6>>; };  // NOLINT
04634     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
04635     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
04636     template<> struct ConwayPolynomial<229, 7> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<223>>; };  // NOLINT
04637     template<> struct ConwayPolynomial<229, 8> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<193>, ZPZV<62>, ZPZV<205>, ZPZV<6>>; };  //
      NOLINT
04638     template<> struct ConwayPolynomial<229, 9> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<117>, ZPZV<50>, ZPZV<223>>;
      };  // NOLINT
04639     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
04640     template<> struct ConwayPolynomial<229, 11> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<2>, ZPZV<223>>; };  // NOLINT
04641     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
04642     template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<223>>; };  // NOLINT
04643     template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<223>>; };  // NOLINT
04644     template<> struct ConwayPolynomial<229, 19> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<228>, ZPZV<15>, ZPZV<223>>; };  //
      NOLINT
04645     template<> struct ConwayPolynomial<233, 1> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<230>>; };  // NOLINT
04646     template<> struct ConwayPolynomial<233, 2> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3>>; };  // NOLINT
04647     template<> struct ConwayPolynomial<233, 3> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<230>>; };  // NOLINT
04648     template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<3>>; };  // NOLINT
04649     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
04650     template<> struct ConwayPolynomial<233, 6> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<122>, ZPZV<215>, ZPZV<32>, ZPZV<3>>; };  // NOLINT
04651     template<> struct ConwayPolynomial<233, 7> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230>>; };  // NOLINT
04652     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>>; };  //
      NOLINT
04653     template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<56>, ZPZV<146>, ZPZV<230>>;
      };  // NOLINT
04654     template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<28>, ZPZV<71>, ZPZV<102>, ZPZV<3>, ZPZV<48>,
```

```
ZPZV<3>>; };  // NOLINT
04655     template<> struct ConwayPolynomial<233, 11> { using ZPZ = aerobus::zpz<233>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<5>, ZPZV<230>>; };  // NOLINT
04656     template<> struct ConwayPolynomial<233, 12> { using ZPZ = aerobus::zpz<233>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<21>, ZPZV<114>, ZPZV<31>, ZPZV<19>,
ZPZV<216>, ZPZV<20>, ZPZV<3>>; };  // NOLINT
04657     template<> struct ConwayPolynomial<233, 13> { using ZPZ = aerobus::zpz<233>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<230>>; };  // NOLINT
04658     template<> struct ConwayPolynomial<233, 17> { using ZPZ = aerobus::zpz<233>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230>>; };  // NOLINT
04659     template<> struct ConwayPolynomial<233, 19> { using ZPZ = aerobus::zpz<233>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<230>>; };  //
NOLINT
04660     template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<232>>; };  // NOLINT
04661     template<> struct ConwayPolynomial<239, 2> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7>>; };  // NOLINT
04662     template<> struct ConwayPolynomial<239, 3> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<232>>; };  // NOLINT
04663     template<> struct ConwayPolynomial<239, 4> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7>>; };  // NOLINT
04664     template<> struct ConwayPolynomial<239, 5> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232>>; };  // NOLINT
04665     template<> struct ConwayPolynomial<239, 6> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<237>, ZPZV<60>, ZPZV<200>, ZPZV<7>>; };  // NOLINT
04666     template<> struct ConwayPolynomial<239, 7> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<232>>; };  // NOLINT
04667     template<> struct ConwayPolynomial<239, 8> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<202>, ZPZV<54>, ZPZV<7>>; };  //
NOLINT
04668     template<> struct ConwayPolynomial<239, 9> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<88>, ZPZV<232>>; };
// NOLINT
04669     template<> struct ConwayPolynomial<239, 10> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<68>, ZPZV<226>, ZPZV<127>,
ZPZV<108>, ZPZV<7>>; };  // NOLINT
04670     template<> struct ConwayPolynomial<239, 11> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<8>, ZPZV<232>>; };  // NOLINT
04671     template<> struct ConwayPolynomial<239, 12> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<235>, ZPZV<14>, ZPZV<113>, ZPZV<182>,
ZPZV<101>, ZPZV<81>, ZPZV<216>, ZPZV<7>>; };  // NOLINT
04672     template<> struct ConwayPolynomial<239, 13> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232>>; };  // NOLINT
04673     template<> struct ConwayPolynomial<239, 17> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<232>>; };  // NOLINT
04674     template<> struct ConwayPolynomial<239, 19> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<232>>; };  //
NOLINT
04675     template<> struct ConwayPolynomial<241, 1> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<234>>; };  // NOLINT
04676     template<> struct ConwayPolynomial<241, 2> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<238>, ZPZV<7>>; };  // NOLINT
04677     template<> struct ConwayPolynomial<241, 3> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<234>>; };  // NOLINT
04678     template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<152>, ZPZV<7>>; };  // NOLINT
04679     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
04680     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
04681     template<> struct ConwayPolynomial<241, 7> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<234>>; };  // NOLINT
04682     template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<173>, ZPZV<212>, ZPZV<153>, ZPZV<7>>; };  //
NOLINT
04683     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<125>, ZPZV<234>>;
};  // NOLINT
04684     template<> struct ConwayPolynomial<241, 10> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<27>, ZPZV<145>, ZPZV<208>, ZPZV<55>,
ZPZV<7>>; };  // NOLINT
04685     template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
ZPZV<3>, ZPZV<234>>; };  // NOLINT
04686     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
04687     template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
```

```
      ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234>>; };  // NOLINT
04688     template<> struct ConwayPolynomial<241, 17> { using ZPZ = aerobus::zpz<241>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<234>>; };  // NOLINT
04689     template<> struct ConwayPolynomial<241, 19> { using ZPZ = aerobus::zpz<241>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234>>; };  //
      NOLINT
04690     template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<245>>; };  // NOLINT
04691     template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6>>; };  // NOLINT
04692     template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<245>>; };  // NOLINT
04693     template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<200>, ZPZV<6>>; };  // NOLINT
04694     template<> struct ConwayPolynomial<251, 5> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<245>>; };  // NOLINT
04695     template<> struct ConwayPolynomial<251, 6> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<247>, ZPZV<151>, ZPZV<179>, ZPZV<6>>; };  // NOLINT
04696     template<> struct ConwayPolynomial<251, 7> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<245>>; };  // NOLINT
04697     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
04698     template<> struct ConwayPolynomial<251, 9> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<106>, ZPZV<245>>;
      };  // NOLINT
04699     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
04700     template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<26>, ZPZV<245>>; };  // NOLINT
04701     template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<192>, ZPZV<53>, ZPZV<20>, ZPZV<20>, ZPZV<15>,
      ZPZV<201>, ZPZV<232>, ZPZV<6>>; };  // NOLINT
04702     template<> struct ConwayPolynomial<251, 13> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<245>>; };  // NOLINT
04703     template<> struct ConwayPolynomial<251, 17> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<245>>; };  // NOLINT
04704     template<> struct ConwayPolynomial<251, 19> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<245>>; };  //
      NOLINT
04705     template<> struct ConwayPolynomial<257, 1> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<254>>; };  // NOLINT
04706     template<> struct ConwayPolynomial<257, 2> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3>>; };  // NOLINT
04707     template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<254>>; };  // NOLINT
04708     template<> struct ConwayPolynomial<257, 4> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<187>, ZPZV<3>>; };  // NOLINT
04709     template<> struct ConwayPolynomial<257, 5> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<254>>; };  // NOLINT
04710     template<> struct ConwayPolynomial<257, 6> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<62>, ZPZV<18>, ZPZV<138>, ZPZV<3>>; };  // NOLINT
04711     template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<254>>; };  // NOLINT
04712     template<> struct ConwayPolynomial<257, 8> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<179>, ZPZV<140>, ZPZV<162>, ZPZV<3>>; };  //
      NOLINT
04713     template<> struct ConwayPolynomial<257, 9> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<50>, ZPZV<254>>;
      };  // NOLINT
04714     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
04715     template<> struct ConwayPolynomial<257, 11> { 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<40>, ZPZV<254>>; };  // NOLINT
04716     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<173>,
      ZPZV<249>, ZPZV<148>, ZPZV<20>, ZPZV<3>>; };  // NOLINT
04717     template<> struct ConwayPolynomial<257, 13> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<254>>; };  // NOLINT
04718     template<> struct ConwayPolynomial<257, 17> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<254>>; };  // NOLINT
04719     template<> struct ConwayPolynomial<257, 19> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<254>>; };  //
      NOLINT
04720     template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<258>>; };  // NOLINT
04721     template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5>>; };  // NOLINT
04722     template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258>>; };  // NOLINT
04723     template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5>>; };  // NOLINT
04724     template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<258>>; };  // NOLINT
04725     template<> struct ConwayPolynomial<263, 6> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<250>, ZPZV<225>, ZPZV<5>>; };  // NOLINT
04726     template<> struct ConwayPolynomial<263, 7> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<258>>; };  // NOLINT
04727     template<> struct ConwayPolynomial<263, 8> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<227>, ZPZV<170>, ZPZV<7>, ZPZV<5>>; };  //
      NOLINT
04728     template<> struct ConwayPolynomial<263, 9> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<261>, ZPZV<29>, ZPZV<258>>;
      };  // NOLINT
04729     template<> struct ConwayPolynomial<263, 10> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<245>, ZPZV<231>, ZPZV<198>, ZPZV<145>,
      ZPZV<119>, ZPZV<5>>; };  // NOLINT
04730     template<> struct ConwayPolynomial<263, 11> { using ZPZ = aerobus::zpz<263>; 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<258>>; };  // NOLINT
04731     template<> struct ConwayPolynomial<263, 12> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<172>, ZPZV<174>, ZPZV<162>, ZPZV<252>,
      ZPZV<47>, ZPZV<45>, ZPZV<180>, ZPZV<5>>; };  // NOLINT
04732     template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<267>>; };  // NOLINT
04733     template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<268>, ZPZV<2>>; };  // NOLINT
04734     template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<267>>; };  // NOLINT
04735     template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2>>; };  // NOLINT
04736     template<> struct ConwayPolynomial<269, 5> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<267>>; };  // NOLINT
04737     template<> struct ConwayPolynomial<269, 6> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<120>, ZPZV<101>, ZPZV<206>, ZPZV<2>>; };  // NOLINT
04738     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<267>>; };  // NOLINT
04739     template<> struct ConwayPolynomial<269, 8> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<220>, ZPZV<131>, ZPZV<232>, ZPZV<2>>; };  //
      NOLINT
04740     template<> struct ConwayPolynomial<269, 9> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<214>, ZPZV<267>, ZPZV<267>>;
      };  // NOLINT
04741     template<> struct ConwayPolynomial<269, 10> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<264>, ZPZV<243>, ZPZV<186>, ZPZV<61>,
      ZPZV<10>, ZPZV<2>>; };  // NOLINT
04742     template<> struct ConwayPolynomial<269, 11> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<20>, ZPZV<267>>; };  // NOLINT
04743     template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<165>, ZPZV<63>, ZPZV<215>,
      ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2>>; };  // NOLINT
04744     template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<265>>; };  // NOLINT
04745     template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6>>; };  // NOLINT
04746     template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<265>>; };  // NOLINT
04747     template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6>>; };  // NOLINT
04748     template<> struct ConwayPolynomial<271, 5> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265>>; };  // NOLINT
04749     template<> struct ConwayPolynomial<271, 6> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<207>, ZPZV<207>, ZPZV<81>, ZPZV<6>>; };  // NOLINT
04750     template<> struct ConwayPolynomial<271, 7> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<265>>; };  // NOLINT
04751     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<6>>; };  //
      NOLINT
04752     template<> struct ConwayPolynomial<271, 9> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<266>, ZPZV<186>,
      ZPZV<265>>; };  // NOLINT
04753     template<> struct ConwayPolynomial<271, 10> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<133>, ZPZV<10>, ZPZV<256>, ZPZV<74>,
      ZPZV<126>, ZPZV<6>>; };  // NOLINT
04754     template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<10>, ZPZV<265>>; };  // NOLINT
04755     template<> struct ConwayPolynomial<271, 12> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<162>, ZPZV<210>, ZPZV<116>, ZPZV<205>,
      ZPZV<237>, ZPZV<256>, ZPZV<130>, ZPZV<6>>; };  // NOLINT
04756     template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<272>>; };  // NOLINT
04757     template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5>>; };  // NOLINT
04758     template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272>>; };  // NOLINT
04759     template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<5>>; };  // NOLINT
04760     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
04761     template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<9>, ZPZV<118>, ZPZV<5>>; };  // NOLINT
04762     template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<272>>; };  // NOLINT
04763     template<> struct ConwayPolynomial<277, 8> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<159>, ZPZV<176>, ZPZV<5>>; };  //
      NOLINT
04764     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
04765     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
04766     template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<5>, ZPZV<272>>; };  // NOLINT
04767     template<> struct ConwayPolynomial<277, 12> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<183>, ZPZV<218>, ZPZV<240>, ZPZV<40>,
      ZPZV<180>, ZPZV<115>, ZPZV<202>, ZPZV<5>>; };  // NOLINT
04768     template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<278>>; };  // NOLINT
04769     template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3>>; };  // NOLINT
04770     template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<278>>; };  // NOLINT
04771     template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3>>; };  // NOLINT
04772     template<> struct ConwayPolynomial<281, 5> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<278>>; };  // NOLINT
04773     template<> struct ConwayPolynomial<281, 6> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<151>, ZPZV<13>, ZPZV<27>, ZPZV<3>>; };  // NOLINT
04774     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<19>, ZPZV<278>>; };  // NOLINT
04775     template<> struct ConwayPolynomial<281, 8> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195>, ZPZV<279>, ZPZV<140>, ZPZV<3>>; };  //
      NOLINT
04776     template<> struct ConwayPolynomial<281, 9> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<70>, ZPZV<278>>;
      };  // NOLINT
04777     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
04778     template<> struct ConwayPolynomial<281, 11> { using ZPZ = aerobus::zpz<281>; 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<278>>; };  // NOLINT
04779     template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<202>, ZPZV<68>, ZPZV<103>, ZPZV<116>,
      ZPZV<58>, ZPZV<28>, ZPZV<191>, ZPZV<3>>; };  // NOLINT
04780     template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<280>>; };  // NOLINT
04781     template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3>>; };  // NOLINT
04782     template<> struct ConwayPolynomial<283, 3> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<280>>; };  // NOLINT
04783     template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3>>; };  // NOLINT
04784     template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<280>>; };  // NOLINT
04785     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
04786     template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<280>>; };  // NOLINT
04787     template<> struct ConwayPolynomial<283, 8> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<179>, ZPZV<32>, ZPZV<232>, ZPZV<3>>; };  //
      NOLINT
04788     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<136>, ZPZV<65>, ZPZV<280>>;
      };  // NOLINT
04789     template<> struct ConwayPolynomial<283, 10> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<271>, ZPZV<185>, ZPZV<68>, ZPZV<100>,
      ZPZV<219>, ZPZV<3>>; };  // NOLINT
04790     template<> struct ConwayPolynomial<283, 11> { 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<0>, ZPZV<0>,
      ZPZV<4>, ZPZV<280>>; };  // NOLINT
04791     template<> struct ConwayPolynomial<283, 12> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<8>, ZPZV<96>, ZPZV<229>, ZPZV<49>,
      ZPZV<14>, ZPZV<56>, ZPZV<3>>; };  // NOLINT
04792     template<> struct ConwayPolynomial<293, 1> { using ZPZ = aerobus::zpz<293>; using type =
```

```
    POLYV<ZPZV<1>, ZPZV<291>>; };  // NOLINT
04793     template<> struct ConwayPolynomial<293, 2> { using ZPZ = aerobus::zpz<293>; using type =
    POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2>>; };  // NOLINT
04794     template<> struct ConwayPolynomial<293, 3> { using ZPZ = aerobus::zpz<293>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291>>; };  // NOLINT
04795     template<> struct ConwayPolynomial<293, 4> { using ZPZ = aerobus::zpz<293>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2>>; };  // NOLINT
04796     template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291>>; };  // NOLINT
04797     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
04798     template<> struct ConwayPolynomial<293, 7> { using ZPZ = aerobus::zpz<293>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<291>>; };  // NOLINT
04799     template<> struct ConwayPolynomial<293, 8> { using ZPZ = aerobus::zpz<293>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<2>>; };  //
    NOLINT
04800     template<> struct ConwayPolynomial<293, 9> { using ZPZ = aerobus::zpz<293>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<208>, ZPZV<190>, ZPZV<291>>;
    };  // NOLINT
04801     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
04802     template<> struct ConwayPolynomial<293, 11> { using ZPZ = aerobus::zpz<293>; 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<291>>; };  // NOLINT
04803     template<> struct ConwayPolynomial<293, 12> { using ZPZ = aerobus::zpz<293>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<159>, ZPZV<210>, ZPZV<125>, ZPZV<212>,
    ZPZV<167>, ZPZV<144>, ZPZV<157>, ZPZV<2>>; };  // NOLINT
04804     template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
    POLYV<ZPZV<1>, ZPZV<302>>; };  // NOLINT
04805     template<> struct ConwayPolynomial<307, 2> { using ZPZ = aerobus::zpz<307>; using type =
    POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5>>; };  // NOLINT
04806     template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<302>>; };  // NOLINT
04807     template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5>>; };  // NOLINT
04808     template<> struct ConwayPolynomial<307, 5> { using ZPZ = aerobus::zpz<307>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<302>>; };  // NOLINT
04809     template<> struct ConwayPolynomial<307, 6> { using ZPZ = aerobus::zpz<307>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<213>, ZPZV<172>, ZPZV<61>, ZPZV<5>>; };  // NOLINT
04810     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<302>>; };  // NOLINT
04811     template<> struct ConwayPolynomial<307, 8> { using ZPZ = aerobus::zpz<307>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<283>, ZPZV<232>, ZPZV<131>, ZPZV<5>>; };  //
    NOLINT
04812     template<> struct ConwayPolynomial<307, 9> { using ZPZ = aerobus::zpz<307>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<165>, ZPZV<70>, ZPZV<302>>;
    };  // NOLINT
04813     template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
    POLYV<ZPZV<1>, ZPZV<294>>; };  // NOLINT
04814     template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
    POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17>>; };  // NOLINT
04815     template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294>>; };  // NOLINT
04816     template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17>>; };  // NOLINT
04817     template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294>>; };  // NOLINT
04818     template<> struct ConwayPolynomial<311, 6> { using ZPZ = aerobus::zpz<311>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<167>, ZPZV<152>, ZPZV<17>>; };  // NOLINT
04819     template<> struct ConwayPolynomial<311, 7> { using ZPZ = aerobus::zpz<311>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<294>>; };  // NOLINT
04820     template<> struct ConwayPolynomial<311, 8> { using ZPZ = aerobus::zpz<311>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<162>, ZPZV<118>, ZPZV<2>, ZPZV<17>>; };  //
    NOLINT
04821     template<> struct ConwayPolynomial<311, 9> { using ZPZ = aerobus::zpz<311>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<287>, ZPZV<74>, ZPZV<294>>;
    };  // NOLINT
04822     template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
    POLYV<ZPZV<1>, ZPZV<303>>; };  // NOLINT
04823     template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
    POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10>>; };  // NOLINT
04824     template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303>>; };  // NOLINT
04825     template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<239>, ZPZV<10>>; };  // NOLINT
04826     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
04827     template<> struct ConwayPolynomial<313, 6> { using ZPZ = aerobus::zpz<313>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<196>, ZPZV<213>, ZPZV<253>, ZPZV<10>>; };  // NOLINT
04828     template<> struct ConwayPolynomial<313, 7> { using ZPZ = aerobus::zpz<313>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<303>>; };  // NOLINT
04829     template<> struct ConwayPolynomial<313, 8> { using ZPZ = aerobus::zpz<313>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<306>, ZPZV<99>, ZPZV<106>, ZPZV<10>>; };  //
    NOLINT
04830     template<> struct ConwayPolynomial<313, 9> { using ZPZ = aerobus::zpz<313>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<267>, ZPZV<300>, ZPZV<303>>;
```

```
      };  // NOLINT
04831     template<> struct ConwayPolynomial<317, 1> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<315>>; };  // NOLINT
04832     template<> struct ConwayPolynomial<317, 2> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2>>; };  // NOLINT
04833     template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<315>>; };  // NOLINT
04834     template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2>>; };  // NOLINT
04835     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
04836     template<> struct ConwayPolynomial<317, 6> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<195>, ZPZV<156>, ZPZV<4>, ZPZV<2>>; };  // NOLINT
04837     template<> struct ConwayPolynomial<317, 7> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<315>>; };  // NOLINT
04838     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
04839     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
04840     template<> struct ConwayPolynomial<331, 1> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<328>>; };  // NOLINT
04841     template<> struct ConwayPolynomial<331, 2> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3>>; };  // NOLINT
04842     template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<328>>; };  // NOLINT
04843     template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<290>, ZPZV<3>>; };  // NOLINT
04844     template<> struct ConwayPolynomial<331, 5> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<328>>; };  // NOLINT
04845     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
04846     template<> struct ConwayPolynomial<331, 7> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<328>>; };  // NOLINT
04847     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
04848     template<> struct ConwayPolynomial<331, 9> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<194>, ZPZV<210>, ZPZV<328>>;
      };  // NOLINT
04849     template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<327>>; };  // NOLINT
04850     template<> struct ConwayPolynomial<337, 2> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<332>, ZPZV<10>>; };  // NOLINT
04851     template<> struct ConwayPolynomial<337, 3> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327>>; };  // NOLINT
04852     template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10>>; };  // NOLINT
04853     template<> struct ConwayPolynomial<337, 5> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<327>>; };  // NOLINT
04854     template<> struct ConwayPolynomial<337, 6> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<216>, ZPZV<127>, ZPZV<109>, ZPZV<10>>; };  // NOLINT
04855     template<> struct ConwayPolynomial<337, 7> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<327>>; };  // NOLINT
04856     template<> struct ConwayPolynomial<337, 8> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<331>, ZPZV<246>, ZPZV<251>, ZPZV<10>>; };  //
      NOLINT
04857     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
04858     template<> struct ConwayPolynomial<347, 1> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<345>>; };  // NOLINT
04859     template<> struct ConwayPolynomial<347, 2> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2>>; };  // NOLINT
04860     template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<345>>; };  // NOLINT
04861     template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2>>; };  // NOLINT
04862     template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<345>>; };  // NOLINT
04863     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
04864     template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<345>>; };  // NOLINT
04865     template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<187>, ZPZV<213>, ZPZV<117>, ZPZV<2>>; };  //
      NOLINT
04866     template<> struct ConwayPolynomial<347, 9> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<235>, ZPZV<252>, ZPZV<345>>;
      };  // NOLINT
04867     template<> struct ConwayPolynomial<349, 1> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<347>>; };  // NOLINT
04868     template<> struct ConwayPolynomial<349, 2> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2>>; };  // NOLINT
04869     template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347>>; };  // NOLINT
```

```
04870     template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2>>; };  // NOLINT
04871     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
04872     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
04873     template<> struct ConwayPolynomial<349, 7> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347>>; };  // NOLINT
04874     template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<328>, ZPZV<268>, ZPZV<2>>; };  //
      NOLINT
04875     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
04876     template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<350>>; };  // NOLINT
04877     template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3>>; };  // NOLINT
04878     template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<350>>; };  // NOLINT
04879     template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<3>>; };  // NOLINT
04880     template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<350>>; };  // NOLINT
04881     template<> struct ConwayPolynomial<353, 6> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<215>, ZPZV<226>, ZPZV<295>, ZPZV<3>>; };  // NOLINT
04882     template<> struct ConwayPolynomial<353, 7> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<350>>; };  // NOLINT
04883     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
04884     template<> struct ConwayPolynomial<353, 9> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<319>, ZPZV<49>, ZPZV<350>>;
      };  // NOLINT
04885     template<> struct ConwayPolynomial<359, 1> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<352>>; };  // NOLINT
04886     template<> struct ConwayPolynomial<359, 2> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7>>; };  // NOLINT
04887     template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<352>>; };  // NOLINT
04888     template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<229>, ZPZV<7>>; };  // NOLINT
04889     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
04890     template<> struct ConwayPolynomial<359, 6> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<309>, ZPZV<327>, ZPZV<327>, ZPZV<7>>; };  // NOLINT
04891     template<> struct ConwayPolynomial<359, 7> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<352>>; };  // NOLINT
04892     template<> struct ConwayPolynomial<359, 8> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<301>, ZPZV<143>, ZPZV<271>, ZPZV<7>>; };  //
      NOLINT
04893     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
04894     template<> struct ConwayPolynomial<367, 1> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<361>>; };  // NOLINT
04895     template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6>>; };  // NOLINT
04896     template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<361>>; };  // NOLINT
04897     template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6>>; };  // NOLINT
04898     template<> struct ConwayPolynomial<367, 5> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<361>>; };  // NOLINT
04899     template<> struct ConwayPolynomial<367, 6> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<222>, ZPZV<321>, ZPZV<324>, ZPZV<6>>; };  // NOLINT
04900     template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<361>>; };  // NOLINT
04901     template<> struct ConwayPolynomial<367, 8> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<335>, ZPZV<282>, ZPZV<50>, ZPZV<6>>; };  //
      NOLINT
04902     template<> struct ConwayPolynomial<367, 9> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<213>, ZPZV<268>,
      ZPZV<361>>; };  // NOLINT
04903     template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<371>>; };  // NOLINT
04904     template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2>>; };  // NOLINT
04905     template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<371>>; };  // NOLINT
04906     template<> struct ConwayPolynomial<373, 4> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304>, ZPZV<2>>; };  // NOLINT
04907     template<> struct ConwayPolynomial<373, 5> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<371>>; };  // NOLINT
04908     template<> struct ConwayPolynomial<373, 6> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<83>, ZPZV<108>, ZPZV<2>>; };  // NOLINT
04909     template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<371>>; };  // NOLINT
04910     template<> struct ConwayPolynomial<373, 8> { using ZPZ = aerobus::zpz<373>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<203>, ZPZV<219>, ZPZV<66>, ZPZV<2>>; };  //
NOLINT
04911     template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<238>, ZPZV<370>,
ZPZV<371>>; };  // NOLINT
04912     template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
POLYV<ZPZV<1>, ZPZV<377>>; };  // NOLINT
04913     template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2>>; };  // NOLINT
04914     template<> struct ConwayPolynomial<379, 3> { using ZPZ = aerobus::zpz<379>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377>>; };  // NOLINT
04915     template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327>, ZPZV<2>>; };  // NOLINT
04916     template<> struct ConwayPolynomial<379, 5> { using ZPZ = aerobus::zpz<379>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<377>>; };  // NOLINT
04917     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
04918     template<> struct ConwayPolynomial<379, 7> { using ZPZ = aerobus::zpz<379>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<377>>; };  // NOLINT
04919     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
04920     template<> struct ConwayPolynomial<379, 9> { using ZPZ = aerobus::zpz<379>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<362>, ZPZV<369>,
ZPZV<377>>; };  // NOLINT
04921     template<> struct ConwayPolynomial<383, 1> { using ZPZ = aerobus::zpz<383>; using type =
POLYV<ZPZV<1>, ZPZV<378>>; };  // NOLINT
04922     template<> struct ConwayPolynomial<383, 2> { using ZPZ = aerobus::zpz<383>; using type =
POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5>>; };  // NOLINT
04923     template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<378>>; };  // NOLINT
04924     template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5>>; };  // NOLINT
04925     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
04926     template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<8>, ZPZV<158>, ZPZV<5>>; };  // NOLINT
04927     template<> struct ConwayPolynomial<383, 7> { using ZPZ = aerobus::zpz<383>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<378>>; };  // NOLINT
04928     template<> struct ConwayPolynomial<383, 8> { using ZPZ = aerobus::zpz<383>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<281>, ZPZV<332>, ZPZV<296>, ZPZV<5>>; };  //
NOLINT
04929     template<> struct ConwayPolynomial<383, 9> { using ZPZ = aerobus::zpz<383>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137>, ZPZV<76>, ZPZV<378>>;
};  // NOLINT
04930     template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<387>>; };  // NOLINT
04931     template<> struct ConwayPolynomial<389, 2> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<379>, ZPZV<2>>; };  // NOLINT
04932     template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387>>; };  // NOLINT
04933     template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<266>, ZPZV<2>>; };  // NOLINT
04934     template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<387>>; };  // NOLINT
04935     template<> struct ConwayPolynomial<389, 6> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<339>, ZPZV<255>, ZPZV<2>>; };  // NOLINT
04936     template<> struct ConwayPolynomial<389, 7> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<387>>; };  // NOLINT
04937     template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<351>, ZPZV<19>, ZPZV<290>, ZPZV<2>>; };  //
NOLINT
04938     template<> struct ConwayPolynomial<389, 9> { using ZPZ = aerobus::zpz<389>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<308>, ZPZV<387>>;
};  // NOLINT
04939     template<> struct ConwayPolynomial<397, 1> { using ZPZ = aerobus::zpz<397>; using type =
POLYV<ZPZV<1>, ZPZV<392>>; };  // NOLINT
04940     template<> struct ConwayPolynomial<397, 2> { using ZPZ = aerobus::zpz<397>; using type =
POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5>>; };  // NOLINT
04941     template<> struct ConwayPolynomial<397, 3> { using ZPZ = aerobus::zpz<397>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<392>>; };  // NOLINT
04942     template<> struct ConwayPolynomial<397, 4> { using ZPZ = aerobus::zpz<397>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5>>; };  // NOLINT
04943     template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392>>; };  // NOLINT
04944     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
04945     template<> struct ConwayPolynomial<397, 7> { using ZPZ = aerobus::zpz<397>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<392>>; };  // NOLINT
04946     template<> struct ConwayPolynomial<397, 8> { using ZPZ = aerobus::zpz<397>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<375>, ZPZV<255>, ZPZV<203>, ZPZV<5>>; };  //
NOLINT
04947     template<> struct ConwayPolynomial<397, 9> { using ZPZ = aerobus::zpz<397>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<166>, ZPZV<252>, ZPZV<392>>;
};  // NOLINT
```

```
04948     template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<398>>; };  // NOLINT
04949     template<> struct ConwayPolynomial<401, 2> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3>>; };  // NOLINT
04950     template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<398>>; };  // NOLINT
04951     template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<3>>; };  // NOLINT
04952     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
04953     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
04954     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
04955     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
04956     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<158>, ZPZV<398>>;
      };  // NOLINT
04957     template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<388>>; };  // NOLINT
04958     template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21>>; };  // NOLINT
04959     template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388>>; };  // NOLINT
04960     template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<407>, ZPZV<21>>; };  // NOLINT
04961     template<> struct ConwayPolynomial<409, 5> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<388>>; };  // NOLINT
04962     template<> struct ConwayPolynomial<409, 6> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<372>, ZPZV<53>, ZPZV<364>, ZPZV<21>>; };  // NOLINT
04963     template<> struct ConwayPolynomial<409, 7> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<388>>; };  // NOLINT
04964     template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<256>, ZPZV<69>, ZPZV<396>, ZPZV<21>>; };  //
      NOLINT
04965     template<> struct ConwayPolynomial<409, 9> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<318>, ZPZV<211>, ZPZV<388>>;
      };  // NOLINT
04966     template<> struct ConwayPolynomial<419, 1> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<417>>; };  // NOLINT
04967     template<> struct ConwayPolynomial<419, 2> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2>>; };  // NOLINT
04968     template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<417>>; };  // NOLINT
04969     template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<373>, ZPZV<2>>; };  // NOLINT
04970     template<> struct ConwayPolynomial<419, 5> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417>>; };  // NOLINT
04971     template<> struct ConwayPolynomial<419, 6> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<411>, ZPZV<33>, ZPZV<257>, ZPZV<2>>; };  // NOLINT
04972     template<> struct ConwayPolynomial<419, 7> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417>>; };  // NOLINT
04973     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
04974     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<93>, ZPZV<386>, ZPZV<417>>;
      };  // NOLINT
04975     template<> struct ConwayPolynomial<421, 1> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<419>>; };  // NOLINT
04976     template<> struct ConwayPolynomial<421, 2> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2>>; };  // NOLINT
04977     template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419>>; };  // NOLINT
04978     template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2>>; };  // NOLINT
04979     template<> struct ConwayPolynomial<421, 5> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<419>>; };  // NOLINT
04980     template<> struct ConwayPolynomial<421, 6> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<111>, ZPZV<342>, ZPZV<41>, ZPZV<2>>; };  // NOLINT
04981     template<> struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<419>>; };  // NOLINT
04982     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>>; };  //
      NOLINT
04983     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>>; };  // NOLINT
04984     template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<424>>; };  // NOLINT
04985     template<> struct ConwayPolynomial<431, 2> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7>>; };  // NOLINT
04986     template<> struct ConwayPolynomial<431, 3> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424>>; };  // NOLINT
04987     template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
```

```
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7>>; };  // NOLINT
04988     template<> struct ConwayPolynomial<431, 5> { using ZPZ = aerobus::zpz<431>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<424>>; };  // NOLINT
04989     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
04990     template<> struct ConwayPolynomial<431, 7> { using ZPZ = aerobus::zpz<431>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424>>; };  // NOLINT
04991     template<> struct ConwayPolynomial<431, 8> { using ZPZ = aerobus::zpz<431>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<243>, ZPZV<286>, ZPZV<115>, ZPZV<7>>; };  //
    NOLINT
04992     template<> struct ConwayPolynomial<431, 9> { using ZPZ = aerobus::zpz<431>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<71>, ZPZV<329>, ZPZV<424>>;
    };  // NOLINT
04993     template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
    POLYV<ZPZV<1>, ZPZV<428>>; };  // NOLINT
04994     template<> struct ConwayPolynomial<433, 2> { using ZPZ = aerobus::zpz<433>; using type =
    POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5>>; };  // NOLINT
04995     template<> struct ConwayPolynomial<433, 3> { using ZPZ = aerobus::zpz<433>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428>>; };  // NOLINT
04996     template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<402>, ZPZV<5>>; };  // NOLINT
04997     template<> struct ConwayPolynomial<433, 5> { using ZPZ = aerobus::zpz<433>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<428>>; };  // NOLINT
04998     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
04999     template<> struct ConwayPolynomial<433, 7> { using ZPZ = aerobus::zpz<433>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<428>>; };  // NOLINT
05000     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
05001     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>>;
    };  // NOLINT
05002     template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
    POLYV<ZPZV<1>, ZPZV<424>>; };  // NOLINT
05003     template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
    POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15>>; };  // NOLINT
05004     template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424>>; };  // NOLINT
05005     template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15>>; };  // NOLINT
05006     template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424>>; };  // NOLINT
05007     template<> struct ConwayPolynomial<439, 6> { using ZPZ = aerobus::zpz<439>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<324>, ZPZV<190>, ZPZV<15>>; };  // NOLINT
05008     template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424>>; };  // NOLINT
05009     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
05010     template<> struct ConwayPolynomial<439, 9> { using ZPZ = aerobus::zpz<439>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<342>, ZPZV<254>,
    ZPZV<424>>; };  // NOLINT
05011     template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
    POLYV<ZPZV<1>, ZPZV<441>>; };  // NOLINT
05012     template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
    POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2>>; };  // NOLINT
05013     template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<441>>; };  // NOLINT
05014     template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2>>; };  // NOLINT
05015     template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<443>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<441>>; };  // NOLINT
05016     template<> struct ConwayPolynomial<443, 6> { using ZPZ = aerobus::zpz<443>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<298>, ZPZV<218>, ZPZV<41>, ZPZV<2>>; };  // NOLINT
05017     template<> struct ConwayPolynomial<443, 7> { using ZPZ = aerobus::zpz<443>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<441>>; };  // NOLINT
05018     template<> struct ConwayPolynomial<443, 8> { using ZPZ = aerobus::zpz<443>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<217>, ZPZV<290>, ZPZV<2>>; };  //
    NOLINT
05019     template<> struct ConwayPolynomial<443, 9> { using ZPZ = aerobus::zpz<443>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<125>, ZPZV<109>, ZPZV<441>>;
    };  // NOLINT
05020     template<> struct ConwayPolynomial<449, 1> { using ZPZ = aerobus::zpz<449>; using type =
    POLYV<ZPZV<1>, ZPZV<446>>; };  // NOLINT
05021     template<> struct ConwayPolynomial<449, 2> { using ZPZ = aerobus::zpz<449>; using type =
    POLYV<ZPZV<1>, ZPZV<444>, ZPZV<3>>; };  // NOLINT
05022     template<> struct ConwayPolynomial<449, 3> { using ZPZ = aerobus::zpz<449>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446>>; };  // NOLINT
05023     template<> struct ConwayPolynomial<449, 4> { using ZPZ = aerobus::zpz<449>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3>>; };  // NOLINT
05024     template<> struct ConwayPolynomial<449, 5> { using ZPZ = aerobus::zpz<449>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<446>>; };  // NOLINT
05025     template<> struct ConwayPolynomial<449, 6> { using ZPZ = aerobus::zpz<449>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<437>, ZPZV<293>, ZPZV<69>, ZPZV<3>>; };  // NOLINT
05026     template<> struct ConwayPolynomial<449, 7> { using ZPZ = aerobus::zpz<449>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<446>>; };  // NOLINT
```

```
05027     template<> struct ConwayPolynomial<449, 8> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361>, ZPZV<348>, ZPZV<124>, ZPZV<3>>; };  //
      NOLINT
05028     template<> struct ConwayPolynomial<449, 9> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<226>, ZPZV<9>, ZPZV<446>>;
      };  // NOLINT
05029     template<> struct ConwayPolynomial<457, 1> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<444>>; };  // NOLINT
05030     template<> struct ConwayPolynomial<457, 2> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<454>, ZPZV<13>>; };  // NOLINT
05031     template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444>>; };  // NOLINT
05032     template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13>>; };  // NOLINT
05033     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
05034     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
05035     template<> struct ConwayPolynomial<457, 7> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<444>>; };  // NOLINT
05036     template<> struct ConwayPolynomial<457, 8> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<365>, ZPZV<296>, ZPZV<412>, ZPZV<13>>; };  //
      NOLINT
05037     template<> struct ConwayPolynomial<457, 9> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<354>, ZPZV<84>, ZPZV<444>>;
      };  // NOLINT
05038     template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<459>>; };  // NOLINT
05039     template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2>>; };  // NOLINT
05040     template<> struct ConwayPolynomial<461, 3> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459>>; };  // NOLINT
05041     template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2>>; };  // NOLINT
05042     template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<459>>; };  // NOLINT
05043     template<> struct ConwayPolynomial<461, 6> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<439>, ZPZV<432>, ZPZV<329>, ZPZV<2>>; };  // NOLINT
05044     template<> struct ConwayPolynomial<461, 7> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<459>>; };  // NOLINT
05045     template<> struct ConwayPolynomial<461, 8> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<449>, ZPZV<321>, ZPZV<2>>; };  //
      NOLINT
05046     template<> struct ConwayPolynomial<461, 9> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<210>, ZPZV<276>, ZPZV<459>>;
      };  // NOLINT
05047     template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<460>>; };  // NOLINT
05048     template<> struct ConwayPolynomial<463, 2> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3>>; };  // NOLINT
05049     template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<460>>; };  // NOLINT
05050     template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3>>; };  // NOLINT
05051     template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<460>>; };  // NOLINT
05052     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
05053     template<> struct ConwayPolynomial<463, 7> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<460>>; };  // NOLINT
05054     template<> struct ConwayPolynomial<463, 8> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<234>, ZPZV<414>, ZPZV<396>, ZPZV<3>>; };  //
      NOLINT
05055     template<> struct ConwayPolynomial<463, 9> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<433>, ZPZV<227>, ZPZV<460>>;
      };  // NOLINT
05056     template<> struct ConwayPolynomial<467, 1> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<465>>; };  // NOLINT
05057     template<> struct ConwayPolynomial<467, 2> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2>>; };  // NOLINT
05058     template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465>>; };  // NOLINT
05059     template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2>>; };  // NOLINT
05060     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
05061     template<> struct ConwayPolynomial<467, 6> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<123>, ZPZV<62>, ZPZV<237>, ZPZV<2>>; };  // NOLINT
05062     template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<465>>; };  // NOLINT
05063     template<> struct ConwayPolynomial<467, 8> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<318>, ZPZV<413>, ZPZV<289>, ZPZV<2>>; };  //
      NOLINT
05064     template<> struct ConwayPolynomial<467, 9> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<397>, ZPZV<447>, ZPZV<465>>;
      };  // NOLINT
05065     template<> struct ConwayPolynomial<479, 1> { using ZPZ = aerobus::zpz<479>; using type =
```

```
    POLYV<ZPZV<1>, ZPZV<466>>; };  // NOLINT
05066     template<> struct ConwayPolynomial<479, 2> { using ZPZ = aerobus::zpz<479>; using type =
    POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13>>; };  // NOLINT
05067     template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466>>; };  // NOLINT
05068     template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13>>; };  // NOLINT
05069     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
05070     template<> struct ConwayPolynomial<479, 6> { using ZPZ = aerobus::zpz<479>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<243>, ZPZV<287>, ZPZV<334>, ZPZV<13>>; };  // NOLINT
05071     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<4>, ZPZV<466>>; };  // NOLINT
05072     template<> struct ConwayPolynomial<479, 8> { using ZPZ = aerobus::zpz<479>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<247>, ZPZV<440>, ZPZV<17>, ZPZV<13>>; };  //
    NOLINT
05073     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>>;
    };  // NOLINT
05074     template<> struct ConwayPolynomial<487, 1> { using ZPZ = aerobus::zpz<487>; using type =
    POLYV<ZPZV<1>, ZPZV<484>>; };  // NOLINT
05075     template<> struct ConwayPolynomial<487, 2> { using ZPZ = aerobus::zpz<487>; using type =
    POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3>>; };  // NOLINT
05076     template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484>>; };  // NOLINT
05077     template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3>>; };  // NOLINT
05078     template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<484>>; };  // NOLINT
05079     template<> struct ConwayPolynomial<487, 6> { using ZPZ = aerobus::zpz<487>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<427>, ZPZV<185>, ZPZV<3>>; };  // NOLINT
05080     template<> struct ConwayPolynomial<487, 7> { using ZPZ = aerobus::zpz<487>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<484>>; };  // NOLINT
05081     template<> struct ConwayPolynomial<487, 8> { using ZPZ = aerobus::zpz<487>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<249>, ZPZV<137>, ZPZV<3>>; };  //
    NOLINT
05082     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<10>, ZPZV<271>, ZPZV<447>,
    ZPZV<484>>; };  // NOLINT
05083     template<> struct ConwayPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
    POLYV<ZPZV<1>, ZPZV<489>>; };  // NOLINT
05084     template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
    POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2>>; };  // NOLINT
05085     template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<489>>; };  // NOLINT
05086     template<> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360>, ZPZV<2>>; };  // NOLINT
05087     template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489>>; };  // NOLINT
05088     template<> struct ConwayPolynomial<491, 6> { using ZPZ = aerobus::zpz<491>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<369>, ZPZV<402>, ZPZV<125>, ZPZV<2>>; };  // NOLINT
05089     template<> struct ConwayPolynomial<491, 7> { using ZPZ = aerobus::zpz<491>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489>>; };  // NOLINT
05090     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
05091     template<> struct ConwayPolynomial<491, 9> { using ZPZ = aerobus::zpz<491>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<453>, ZPZV<489>>;
    };  // NOLINT
05092     template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
    POLYV<ZPZV<1>, ZPZV<492>>; };  // NOLINT
05093     template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
    POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7>>; };  // NOLINT
05094     template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492>>; };  // NOLINT
05095     template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7>>; };  // NOLINT
05096     template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<492>>; };  // NOLINT
05097     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
05098     template<> struct ConwayPolynomial<499, 7> { using ZPZ = aerobus::zpz<499>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<492>>; };  // NOLINT
05099     template<> struct ConwayPolynomial<499, 8> { using ZPZ = aerobus::zpz<499>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<288>, ZPZV<309>, ZPZV<200>, ZPZV<7>>; };  //
    NOLINT
05100     template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<491>, ZPZV<222>, ZPZV<492>>;
    };  // NOLINT
05101     template<> struct ConwayPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
    POLYV<ZPZV<1>, ZPZV<498>>; };  // NOLINT
05102     template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
    POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5>>; };  // NOLINT
05103     template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<498>>; };  // NOLINT
05104     template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<325>, ZPZV<5>>; };  // NOLINT
```

```
05105     template<> struct ConwayPolynomial<503, 5> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<498>>; };  // NOLINT
05106     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
05107     template<> struct ConwayPolynomial<503, 7> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<498>>; };  // NOLINT
05108     template<> struct ConwayPolynomial<503, 8> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<441>, ZPZV<203>, ZPZV<316>, ZPZV<5>>; };  //
      NOLINT
05109     template<> struct ConwayPolynomial<503, 9> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<158>, ZPZV<337>, ZPZV<498>>;
      };  // NOLINT
05110     template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<507>>; };  // NOLINT
05111     template<> struct ConwayPolynomial<509, 2> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2>>; };  // NOLINT
05112     template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<507>>; };  // NOLINT
05113     template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2>>; };  // NOLINT
05114     template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<507>>; };  // NOLINT
05115     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
05116     template<> struct ConwayPolynomial<509, 7> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<507>>; };  // NOLINT
05117     template<> struct ConwayPolynomial<509, 8> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<420>, ZPZV<473>, ZPZV<382>, ZPZV<2>>; };  //
      NOLINT
05118     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<314>, ZPZV<28>, ZPZV<507>>;
      };  // NOLINT
05119     template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<518>>; };  // NOLINT
05120     template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3>>; };  // NOLINT
05121     template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518>>; };  // NOLINT
05122     template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<509>, ZPZV<3>>; };  // NOLINT
05123     template<> struct ConwayPolynomial<521, 5> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<518>>; };  // NOLINT
05124     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
05125     template<> struct ConwayPolynomial<521, 7> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<518>>; };  // NOLINT
05126     template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<462>, ZPZV<407>, ZPZV<312>, ZPZV<3>>; };  //
      NOLINT
05127     template<> struct ConwayPolynomial<521, 9> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<181>, ZPZV<483>, ZPZV<518>>;
      };  // NOLINT
05128     template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<521>>; };  // NOLINT
05129     template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2>>; };  // NOLINT
05130     template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<521>>; };  // NOLINT
05131     template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2>>; };  // NOLINT
05132     template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<521>>; };  // NOLINT
05133     template<> struct ConwayPolynomial<523, 6> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<475>, ZPZV<475>, ZPZV<371>, ZPZV<2>>; };  // NOLINT
05134     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<13>, ZPZV<521>>; };  // NOLINT
05135     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
05136     template<> struct ConwayPolynomial<523, 9> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<342>, ZPZV<145>,
      ZPZV<521>>; };  // NOLINT
05137     template<> struct ConwayPolynomial<541, 1> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<539>>; };  // NOLINT
05138     template<> struct ConwayPolynomial<541, 2> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2>>; };  // NOLINT
05139     template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539>>; };  // NOLINT
05140     template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2>>; };  // NOLINT
05141     template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<539>>; };  // NOLINT
05142     template<> struct ConwayPolynomial<541, 6> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<239>, ZPZV<320>, ZPZV<69>, ZPZV<2>>; };  // NOLINT
05143     template<> struct ConwayPolynomial<541, 7> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<539>>; };  // NOLINT
05144     template<> struct ConwayPolynomial<541, 8> { using ZPZ = aerobus::zpz<541>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<376>, ZPZV<108>, ZPZV<113>, ZPZV<2>>; };  //
      NOLINT
05145     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
05146     template<> struct ConwayPolynomial<547, 1> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<545>>; };  // NOLINT
05147     template<> struct ConwayPolynomial<547, 2> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2>>; };  // NOLINT
05148     template<> struct ConwayPolynomial<547, 3> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<545>>; };  // NOLINT
05149     template<> struct ConwayPolynomial<547, 4> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<334>, ZPZV<2>>; };  // NOLINT
05150     template<> struct ConwayPolynomial<547, 5> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<545>>; };  // NOLINT
05151     template<> struct ConwayPolynomial<547, 6> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<2>>; };  // NOLINT
05152     template<> struct ConwayPolynomial<547, 7> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<545>>; };  // NOLINT
05153     template<> struct ConwayPolynomial<547, 8> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2>>; };  //
      NOLINT
05154     template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<238>, ZPZV<263>,
      ZPZV<545>>; };  // NOLINT
05155     template<> struct ConwayPolynomial<557, 1> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<555>>; };  // NOLINT
05156     template<> struct ConwayPolynomial<557, 2> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2>>; };  // NOLINT
05157     template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<555>>; };  // NOLINT
05158     template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2>>; };  // NOLINT
05159     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
05160     template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<202>, ZPZV<192>, ZPZV<253>, ZPZV<2>>; };  // NOLINT
05161     template<> struct ConwayPolynomial<557, 7> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<555>>; };  // NOLINT
05162     template<> struct ConwayPolynomial<557, 8> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<480>, ZPZV<384>, ZPZV<113>, ZPZV<2>>; };  //
      NOLINT
05163     template<> struct ConwayPolynomial<557, 9> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<456>, ZPZV<434>, ZPZV<555>>;
      };  // NOLINT
05164     template<> struct ConwayPolynomial<563, 1> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<561>>; };  // NOLINT
05165     template<> struct ConwayPolynomial<563, 2> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2>>; };  // NOLINT
05166     template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561>>; };  // NOLINT
05167     template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<20>, ZPZV<399>, ZPZV<2>>; };  // NOLINT
05168     template<> struct ConwayPolynomial<563, 5> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<561>>; };  // NOLINT
05169     template<> struct ConwayPolynomial<563, 6> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<122>, ZPZV<303>, ZPZV<246>, ZPZV<2>>; };  // NOLINT
05170     template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<561>>; };  // NOLINT
05171     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
05172     template<> struct ConwayPolynomial<563, 9> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<15>, ZPZV<19>, ZPZV<561>>;
      };  // NOLINT
05173     template<> struct ConwayPolynomial<569, 1> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<566>>; };  // NOLINT
05174     template<> struct ConwayPolynomial<569, 2> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3>>; };  // NOLINT
05175     template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566>>; };  // NOLINT
05176     template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3>>; };  // NOLINT
05177     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
05178     template<> struct ConwayPolynomial<569, 6> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<50>, ZPZV<263>, ZPZV<480>, ZPZV<3>>; };  // NOLINT
05179     template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<566>>; };  // NOLINT
05180     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>>; };  //
      NOLINT
05181     template<> struct ConwayPolynomial<569, 9> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<478>, ZPZV<566>, ZPZV<566>>;
      };  // NOLINT
05182     template<> struct ConwayPolynomial<571, 1> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<568>>; };  // NOLINT
```

```
05183     template<> struct ConwayPolynomial<571, 2> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3>>; };  // NOLINT
05184     template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<568>>; };  // NOLINT
05185     template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3>>; };  // NOLINT
05186     template<> struct ConwayPolynomial<571, 5> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<568>>; };  // NOLINT
05187     template<> struct ConwayPolynomial<571, 6> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<221>, ZPZV<295>, ZPZV<33>, ZPZV<3>>; };  // NOLINT
05188     template<> struct ConwayPolynomial<571, 7> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<568>>; };  // NOLINT
05189     template<> struct ConwayPolynomial<571, 8> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<363>, ZPZV<119>, ZPZV<371>, ZPZV<3>>; };  //
      NOLINT
05190     template<> struct ConwayPolynomial<571, 9> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<545>, ZPZV<179>,
      ZPZV<568>>; };  // NOLINT
05191     template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<572>>; };  // NOLINT
05192     template<> struct ConwayPolynomial<577, 2> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5>>; };  // NOLINT
05193     template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572>>; };  // NOLINT
05194     template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<5>>; };  // NOLINT
05195     template<> struct ConwayPolynomial<577, 5> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<572>>; };  // NOLINT
05196     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
05197     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<8>, ZPZV<572>>; };  // NOLINT
05198     template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<5>>; };  //
      NOLINT
05199     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
05200     template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<585>>; };  // NOLINT
05201     template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2>>; };  // NOLINT
05202     template<> struct ConwayPolynomial<587, 3> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585>>; };  // NOLINT
05203     template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<444>, ZPZV<2>>; };  // NOLINT
05204     template<> struct ConwayPolynomial<587, 5> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<585>>; };  // NOLINT
05205     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
05206     template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<585>>; };  // NOLINT
05207     template<> struct ConwayPolynomial<587, 8> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<492>, ZPZV<44>, ZPZV<91>, ZPZV<2>>; };  //
      NOLINT
05208     template<> struct ConwayPolynomial<587, 9> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<333>, ZPZV<55>, ZPZV<585>>;
      };  // NOLINT
05209     template<> struct ConwayPolynomial<593, 1> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<590>>; };  // NOLINT
05210     template<> struct ConwayPolynomial<593, 2> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3>>; };  // NOLINT
05211     template<> struct ConwayPolynomial<593, 3> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<590>>; };  // NOLINT
05212     template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3>>; };  // NOLINT
05213     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
05214     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
05215     template<> struct ConwayPolynomial<593, 7> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<590>>; };  // NOLINT
05216     template<> struct ConwayPolynomial<593, 8> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<350>, ZPZV<291>, ZPZV<495>, ZPZV<3>>; };  //
      NOLINT
05217     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
05218     template<> struct ConwayPolynomial<599, 1> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<592>>; };  // NOLINT
05219     template<> struct ConwayPolynomial<599, 2> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7>>; };  // NOLINT
05220     template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<592>>; };  // NOLINT
05221     template<> struct ConwayPolynomial<599, 4> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7>>; };  // NOLINT
05222     template<> struct ConwayPolynomial<599, 5> { using ZPZ = aerobus::zpz<599>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<592>>; };  // NOLINT
05223     template<> struct ConwayPolynomial<599, 6> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7>>; };  // NOLINT
05224     template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592>>; };  // NOLINT
05225     template<> struct ConwayPolynomial<599, 8> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<440>, ZPZV<37>, ZPZV<124>, ZPZV<7>>; };  //
      NOLINT
05226     template<> struct ConwayPolynomial<599, 9> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<114>, ZPZV<98>, ZPZV<592>>;
      };  // NOLINT
05227     template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<594>>; };  // NOLINT
05228     template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7>>; };  // NOLINT
05229     template<> struct ConwayPolynomial<601, 3> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<594>>; };  // NOLINT
05230     template<> struct ConwayPolynomial<601, 4> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7>>; };  // NOLINT
05231     template<> struct ConwayPolynomial<601, 5> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<594>>; };  // NOLINT
05232     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
05233     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<594>>; };  // NOLINT
05234     template<> struct ConwayPolynomial<601, 8> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<550>, ZPZV<241>, ZPZV<490>, ZPZV<7>>; };  //
      NOLINT
05235     template<> struct ConwayPolynomial<601, 9> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<487>, ZPZV<590>, ZPZV<594>>;
      };  // NOLINT
05236     template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<604>>; };  // NOLINT
05237     template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3>>; };  // NOLINT
05238     template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<604>>; };  // NOLINT
05239     template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3>>; };  // NOLINT
05240     template<> struct ConwayPolynomial<607, 5> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<604>>; };  // NOLINT
05241     template<> struct ConwayPolynomial<607, 6> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<45>, ZPZV<478>, ZPZV<3>>; };  // NOLINT
05242     template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<604>>; };  // NOLINT
05243     template<> struct ConwayPolynomial<607, 8> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<468>, ZPZV<35>, ZPZV<449>, ZPZV<3>>; };  //
      NOLINT
05244     template<> struct ConwayPolynomial<607, 9> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<444>, ZPZV<129>, ZPZV<604>>;
      };  // NOLINT
05245     template<> struct ConwayPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<611>>; };  // NOLINT
05246     template<> struct ConwayPolynomial<613, 2> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2>>; };  // NOLINT
05247     template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<611>>; };  // NOLINT
05248     template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2>>; };  // NOLINT
05249     template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<611>>; };  // NOLINT
05250     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
05251     template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<611>>; };  // NOLINT
05252     template<> struct ConwayPolynomial<613, 8> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<489>, ZPZV<57>, ZPZV<539>, ZPZV<2>>; };  //
      NOLINT
05253     template<> struct ConwayPolynomial<613, 9> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<513>, ZPZV<536>, ZPZV<611>>;
      };  // NOLINT
05254     template<> struct ConwayPolynomial<617, 1> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<614>>; };  // NOLINT
05255     template<> struct ConwayPolynomial<617, 2> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3>>; };  // NOLINT
05256     template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<614>>; };  // NOLINT
05257     template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3>>; };  // NOLINT
05258     template<> struct ConwayPolynomial<617, 5> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<614>>; };  // NOLINT
05259     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
05260     template<> struct ConwayPolynomial<617, 7> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<614>>; };  // NOLINT
05261     template<> struct ConwayPolynomial<617, 8> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<519>, ZPZV<501>, ZPZV<155>, ZPZV<3>>; };  //
```

```
    NOLINT
05262     template<> struct ConwayPolynomial<617, 9> { using ZPZ = aerobus::zpz<617>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<543>,
    ZPZV<614>>; };  // NOLINT
05263     template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
    POLYV<ZPZV<1>, ZPZV<617>>; };  // NOLINT
05264     template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
    POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2>>; };  // NOLINT
05265     template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617>>; };  // NOLINT
05266     template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492>, ZPZV<2>>; };  // NOLINT
05267     template<> struct ConwayPolynomial<619, 5> { using ZPZ = aerobus::zpz<619>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<617>>; };  // NOLINT
05268     template<> struct ConwayPolynomial<619, 6> { using ZPZ = aerobus::zpz<619>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<468>, ZPZV<347>, ZPZV<2>>; };  // NOLINT
05269     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<7>, ZPZV<617>>; };  // NOLINT
05270     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
05271     template<> struct ConwayPolynomial<619, 9> { using ZPZ = aerobus::zpz<619>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<310>, ZPZV<617>>;
    };  // NOLINT
05272     template<> struct ConwayPolynomial<631, 1> { using ZPZ = aerobus::zpz<631>; using type =
    POLYV<ZPZV<1>, ZPZV<628>>; };  // NOLINT
05273     template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
    POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3>>; };  // NOLINT
05274     template<> struct ConwayPolynomial<631, 3> { using ZPZ = aerobus::zpz<631>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<628>>; };  // NOLINT
05275     template<> struct ConwayPolynomial<631, 4> { using ZPZ = aerobus::zpz<631>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<3>>; };  // NOLINT
05276     template<> struct ConwayPolynomial<631, 5> { using ZPZ = aerobus::zpz<631>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628>>; };  // NOLINT
05277     template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<516>, ZPZV<541>, ZPZV<106>, ZPZV<3>>; };  // NOLINT
05278     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<5>, ZPZV<628>>; };  // NOLINT
05279     template<> struct ConwayPolynomial<631, 8> { using ZPZ = aerobus::zpz<631>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<3>>; };  //
    NOLINT
05280     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
05281     template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
    POLYV<ZPZV<1>, ZPZV<638>>; };  // NOLINT
05282     template<> struct ConwayPolynomial<641, 2> { using ZPZ = aerobus::zpz<641>; using type =
    POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3>>; };  // NOLINT
05283     template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<638>>; };  // NOLINT
05284     template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<3>>; };  // NOLINT
05285     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
05286     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
05287     template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<638>>; };  // NOLINT
05288     template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<3>>; };  //
    NOLINT
05289     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>>;
    };  // NOLINT
05290     template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
    POLYV<ZPZV<1>, ZPZV<632>>; };  // NOLINT
05291     template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
    POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11>>; };  // NOLINT
05292     template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<632>>; };  // NOLINT
05293     template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<600>, ZPZV<11>>; };  // NOLINT
05294     template<> struct ConwayPolynomial<643, 5> { using ZPZ = aerobus::zpz<643>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<632>>; };  // NOLINT
05295     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
05296     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
05297     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
05298     template<> struct ConwayPolynomial<643, 9> { using ZPZ = aerobus::zpz<643>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<591>, ZPZV<475>, ZPZV<632>>;
    };  // NOLINT
05299     template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
    POLYV<ZPZV<1>, ZPZV<642>>; };  // NOLINT
05300     template<> struct ConwayPolynomial<647, 2> { using ZPZ = aerobus::zpz<647>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5>>; };  // NOLINT
05301     template<> struct ConwayPolynomial<647, 3> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<642>>; };  // NOLINT
05302     template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5>>; };  // NOLINT
05303     template<> struct ConwayPolynomial<647, 5> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<642>>; };  // NOLINT
05304     template<> struct ConwayPolynomial<647, 6> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<308>, ZPZV<385>, ZPZV<642>, ZPZV<5>>; };  // NOLINT
05305     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
05306     template<> struct ConwayPolynomial<647, 8> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<603>, ZPZV<259>, ZPZV<271>, ZPZV<5>>; };  //
      NOLINT
05307     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
05308     template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<651>>; };  // NOLINT
05309     template<> struct ConwayPolynomial<653, 2> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2>>; };  // NOLINT
05310     template<> struct ConwayPolynomial<653, 3> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<651>>; };  // NOLINT
05311     template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2>>; };  // NOLINT
05312     template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651>>; };  // NOLINT
05313     template<> struct ConwayPolynomial<653, 6> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<45>, ZPZV<220>, ZPZV<242>, ZPZV<2>>; };  // NOLINT
05314     template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<651>>; };  // NOLINT
05315     template<> struct ConwayPolynomial<653, 8> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<385>, ZPZV<18>, ZPZV<296>, ZPZV<2>>; };  //
      NOLINT
05316     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<365>, ZPZV<60>, ZPZV<651>>;
      };  // NOLINT
05317     template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<657>>; };  // NOLINT
05318     template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2>>; };  // NOLINT
05319     template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<657>>; };  // NOLINT
05320     template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2>>; };  // NOLINT
05321     template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<657>>; };  // NOLINT
05322     template<> struct ConwayPolynomial<659, 6> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<371>, ZPZV<105>, ZPZV<223>, ZPZV<2>>; };  // NOLINT
05323     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
05324     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>>; };  //
      NOLINT
05325     template<> struct ConwayPolynomial<659, 9> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592>, ZPZV<46>, ZPZV<657>>;
      };  // NOLINT
05326     template<> struct ConwayPolynomial<661, 1> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<659>>; };  // NOLINT
05327     template<> struct ConwayPolynomial<661, 2> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2>>; };  // NOLINT
05328     template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<659>>; };  // NOLINT
05329     template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2>>; };  // NOLINT
05330     template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<659>>; };  // NOLINT
05331     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
05332     template<> struct ConwayPolynomial<661, 7> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<659>>; };  // NOLINT
05333     template<> struct ConwayPolynomial<661, 8> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<612>, ZPZV<285>, ZPZV<72>, ZPZV<2>>; };  //
      NOLINT
05334     template<> struct ConwayPolynomial<661, 9> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<389>, ZPZV<220>,
      ZPZV<659>>; };  // NOLINT
05335     template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<668>>; };  // NOLINT
05336     template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5>>; };  // NOLINT
05337     template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<668>>; };  // NOLINT
05338     template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5>>; };  // NOLINT
05339     template<> struct ConwayPolynomial<673, 5> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<668>>; };  // NOLINT
```

```
05340     template<> struct ConwayPolynomial<673, 6> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<524>, ZPZV<248>, ZPZV<35>, ZPZV<5>>; };  // NOLINT
05341     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<668>>; };  // NOLINT
05342     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
05343     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
05344     template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<675>>; };  // NOLINT
05345     template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<672>, ZPZV<2>>; };  // NOLINT
05346     template<> struct ConwayPolynomial<677, 3> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<675>>; };  // NOLINT
05347     template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2>>; };  // NOLINT
05348     template<> struct ConwayPolynomial<677, 5> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<675>>; };  // NOLINT
05349     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
05350     template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<675>>; };  // NOLINT
05351     template<> struct ConwayPolynomial<677, 8> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<363>, ZPZV<619>, ZPZV<152>, ZPZV<2>>; };  //
      NOLINT
05352     template<> struct ConwayPolynomial<677, 9> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<504>, ZPZV<404>, ZPZV<675>>;
      };  // NOLINT
05353     template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<678>>; };  // NOLINT
05354     template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5>>; };  // NOLINT
05355     template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<678>>; };  // NOLINT
05356     template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5>>; };  // NOLINT
05357     template<> struct ConwayPolynomial<683, 5> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<678>>; };  // NOLINT
05358     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
05359     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<30>, ZPZV<678>>; };  // NOLINT
05360     template<> struct ConwayPolynomial<683, 8> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<383>, ZPZV<184>, ZPZV<65>, ZPZV<5>>; };  //
      NOLINT
05361     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<85>, ZPZV<444>, ZPZV<678>>;
      };  // NOLINT
05362     template<> struct ConwayPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<688>>; };  // NOLINT
05363     template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3>>; };  // NOLINT
05364     template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<688>>; };  // NOLINT
05365     template<> struct ConwayPolynomial<691, 4> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3>>; };  // NOLINT
05366     template<> struct ConwayPolynomial<691, 5> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688>>; };  // NOLINT
05367     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
05368     template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688>>; };  // NOLINT
05369     template<> struct ConwayPolynomial<691, 8> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<356>, ZPZV<425>, ZPZV<321>, ZPZV<3>>; };  //
      NOLINT
05370     template<> struct ConwayPolynomial<691, 9> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<556>, ZPZV<443>, ZPZV<688>>;
      };  // NOLINT
05371     template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<699>>; };  // NOLINT
05372     template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2>>; };  // NOLINT
05373     template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699>>; };  // NOLINT
05374     template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2>>; };  // NOLINT
05375     template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699>>; };  // NOLINT
05376     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
05377     template<> struct ConwayPolynomial<701, 7> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<699>>; };  // NOLINT
05378     template<> struct ConwayPolynomial<701, 8> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<619>, ZPZV<206>, ZPZV<593>, ZPZV<2>>; };  //
      NOLINT
```

```
05379     template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<459>, ZPZV<373>, ZPZV<699>>;
      };  // NOLINT
05380     template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<707>>; };  // NOLINT
05381     template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2>>; };  // NOLINT
05382     template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707>>; };  // NOLINT
05383     template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2>>; };  // NOLINT
05384     template<> struct ConwayPolynomial<709, 5> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<707>>; };  // NOLINT
05385     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
05386     template<> struct ConwayPolynomial<709, 7> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<707>>; };  // NOLINT
05387     template<> struct ConwayPolynomial<709, 8> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<689>, ZPZV<233>, ZPZV<79>, ZPZV<2>>; };  //
      NOLINT
05388     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
05389     template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<708>>; };  // NOLINT
05390     template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11>>; };  // NOLINT
05391     template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708>>; };  // NOLINT
05392     template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<602>, ZPZV<11>>; };  // NOLINT
05393     template<> struct ConwayPolynomial<719, 5> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708>>; };  // NOLINT
05394     template<> struct ConwayPolynomial<719, 6> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<533>, ZPZV<591>, ZPZV<182>, ZPZV<11>>; };  // NOLINT
05395     template<> struct ConwayPolynomial<719, 7> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<708>>; };  // NOLINT
05396     template<> struct ConwayPolynomial<719, 8> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<714>, ZPZV<362>, ZPZV<244>, ZPZV<11>>; };  //
      NOLINT
05397     template<> struct ConwayPolynomial<719, 9> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<288>, ZPZV<560>, ZPZV<708>>;
      };  // NOLINT
05398     template<> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<722>>; };  // NOLINT
05399     template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5>>; };  // NOLINT
05400     template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<722>>; };  // NOLINT
05401     template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5>>; };  // NOLINT
05402     template<> struct ConwayPolynomial<727, 5> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<722>>; };  // NOLINT
05403     template<> struct ConwayPolynomial<727, 6> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<86>, ZPZV<397>, ZPZV<672>, ZPZV<5>>; };  // NOLINT
05404     template<> struct ConwayPolynomial<727, 7> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<722>>; };  // NOLINT
05405     template<> struct ConwayPolynomial<727, 8> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<639>, ZPZV<671>, ZPZV<368>, ZPZV<5>>; };  //
      NOLINT
05406     template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<573>, ZPZV<502>, ZPZV<722>>;
      };  // NOLINT
05407     template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<727>>; };  // NOLINT
05408     template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6>>; };  // NOLINT
05409     template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727>>; };  // NOLINT
05410     template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<6>>; };  // NOLINT
05411     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
05412     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
05413     template<> struct ConwayPolynomial<733, 7> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<727>>; };  // NOLINT
05414     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
05415     template<> struct ConwayPolynomial<733, 9> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<337>, ZPZV<6>, ZPZV<727>>;
      };  // NOLINT
05416     template<> struct ConwayPolynomial<739, 1> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<736>>; };  // NOLINT
05417     template<> struct ConwayPolynomial<739, 2> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3>>; };  // NOLINT
```

```
05418     template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736>>; };  // NOLINT
05419     template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<3>>; };  // NOLINT
05420     template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<736>>; };  // NOLINT
05421     template<> struct ConwayPolynomial<739, 6> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<422>, ZPZV<447>, ZPZV<625>, ZPZV<3>>; };  // NOLINT
05422     template<> struct ConwayPolynomial<739, 7> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<44>, ZPZV<736>>; };  // NOLINT
05423     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
05424     template<> struct ConwayPolynomial<739, 9> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<616>, ZPZV<81>, ZPZV<736>>;
      };  // NOLINT
05425     template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<738>>; };  // NOLINT
05426     template<> struct ConwayPolynomial<743, 2> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5>>; };  // NOLINT
05427     template<> struct ConwayPolynomial<743, 3> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<738>>; };  // NOLINT
05428     template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<5>>; };  // NOLINT
05429     template<> struct ConwayPolynomial<743, 5> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738>>; };  // NOLINT
05430     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
05431     template<> struct ConwayPolynomial<743, 7> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738>>; };  // NOLINT
05432     template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<279>, ZPZV<588>, ZPZV<5>>; };  //
      NOLINT
05433     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
05434     template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<748>>; };  // NOLINT
05435     template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3>>; };  // NOLINT
05436     template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748>>; };  // NOLINT
05437     template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<525>, ZPZV<3>>; };  // NOLINT
05438     template<> struct ConwayPolynomial<751, 5> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748>>; };  // NOLINT
05439     template<> struct ConwayPolynomial<751, 6> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<298>, ZPZV<633>, ZPZV<539>, ZPZV<3>>; };  // NOLINT
05440     template<> struct ConwayPolynomial<751, 7> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748>>; };  // NOLINT
05441     template<> struct ConwayPolynomial<751, 8> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<741>, ZPZV<243>, ZPZV<672>, ZPZV<3>>; };  //
      NOLINT
05442     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
05443     template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<755>>; };  // NOLINT
05444     template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2>>; };  // NOLINT
05445     template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<755>>; };  // NOLINT
05446     template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2>>; };  // NOLINT
05447     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
05448     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
05449     template<> struct ConwayPolynomial<757, 7> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<755>>; };  // NOLINT
05450     template<> struct ConwayPolynomial<757, 8> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<110>, ZPZV<509>, ZPZV<2>>; };  //
      NOLINT
05451     template<> struct ConwayPolynomial<757, 9> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<688>, ZPZV<702>, ZPZV<755>>;
      };  // NOLINT
05452     template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<755>>; };  // NOLINT
05453     template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6>>; };  // NOLINT
05454     template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755>>; };  // NOLINT
05455     template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<6>>; };  // NOLINT
05456     template<> struct ConwayPolynomial<761, 5> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<755>>; };  // NOLINT
05457     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
05458     template<> struct ConwayPolynomial<761, 7> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<755>>; };  // NOLINT
05459     template<> struct ConwayPolynomial<761, 8> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<603>, ZPZV<144>, ZPZV<540>, ZPZV<6>>; };  //
      NOLINT
05460     template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<317>, ZPZV<571>, ZPZV<755>>;
      };  // NOLINT
05461     template<> struct ConwayPolynomial<769, 1> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<758>>; };  // NOLINT
05462     template<> struct ConwayPolynomial<769, 2> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11>>; };  // NOLINT
05463     template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758>>; };  // NOLINT
05464     template<> struct ConwayPolynomial<769, 4> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11>>; };  // NOLINT
05465     template<> struct ConwayPolynomial<769, 5> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<758>>; };  // NOLINT
05466     template<> struct ConwayPolynomial<769, 6> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<326>, ZPZV<650>, ZPZV<11>>; };  // NOLINT
05467     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<758>>; };  // NOLINT
05468     template<> struct ConwayPolynomial<769, 8> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<560>, ZPZV<574>, ZPZV<632>, ZPZV<11>>; };  //
      NOLINT
05469     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<623>, ZPZV<751>, ZPZV<758>>;
      };  // NOLINT
05470     template<> struct ConwayPolynomial<773, 1> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<771>>; };  // NOLINT
05471     template<> struct ConwayPolynomial<773, 2> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2>>; };  // NOLINT
05472     template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<771>>; };  // NOLINT
05473     template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2>>; };  // NOLINT
05474     template<> struct ConwayPolynomial<773, 5> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<771>>; };  // NOLINT
05475     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
05476     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<9>, ZPZV<771>>; };  // NOLINT
05477     template<> struct ConwayPolynomial<773, 8> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<484>, ZPZV<94>, ZPZV<693>, ZPZV<2>>; };  //
      NOLINT
05478     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
05479     template<> struct ConwayPolynomial<787, 1> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<785>>; };  // NOLINT
05480     template<> struct ConwayPolynomial<787, 2> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2>>; };  // NOLINT
05481     template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785>>; };  // NOLINT
05482     template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2>>; };  // NOLINT
05483     template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<785>>; };  // NOLINT
05484     template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<98>, ZPZV<512>, ZPZV<606>, ZPZV<2>>; };  // NOLINT
05485     template<> struct ConwayPolynomial<787, 7> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<785>>; };  // NOLINT
05486     template<> struct ConwayPolynomial<787, 8> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<612>, ZPZV<26>, ZPZV<715>, ZPZV<2>>; };  //
      NOLINT
05487     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
05488     template<> struct ConwayPolynomial<797, 1> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<795>>; };  // NOLINT
05489     template<> struct ConwayPolynomial<797, 2> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2>>; };  // NOLINT
05490     template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<795>>; };  // NOLINT
05491     template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<717>, ZPZV<2>>; };  // NOLINT
05492     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
05493     template<> struct ConwayPolynomial<797, 6> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<657>, ZPZV<396>, ZPZV<71>, ZPZV<2>>; };  // NOLINT
05494     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<11>, ZPZV<795>>; };  // NOLINT
05495     template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2>>; };  //
      NOLINT
05496     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<599>, ZPZV<795>>;
      };  // NOLINT
05497     template<> struct ConwayPolynomial<809, 1> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<806>>; };  // NOLINT
05498     template<> struct ConwayPolynomial<809, 2> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3>>; };  // NOLINT
05499     template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<806>>; };  // NOLINT
05500     template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3>>; };  // NOLINT
05501     template<> struct ConwayPolynomial<809, 5> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806>>; };  // NOLINT
05502     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
05503     template<> struct ConwayPolynomial<809, 7> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806>>; };  // NOLINT
05504     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
05505     template<> struct ConwayPolynomial<809, 9> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<341>, ZPZV<727>, ZPZV<806>>;
      };  // NOLINT
05506     template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<808>>; };  // NOLINT
05507     template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<806>, ZPZV<3>>; };  // NOLINT
05508     template<> struct ConwayPolynomial<811, 3> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<808>>; };  // NOLINT
05509     template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3>>; };  // NOLINT
05510     template<> struct ConwayPolynomial<811, 5> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<808>>; };  // NOLINT
05511     template<> struct ConwayPolynomial<811, 6> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<780>, ZPZV<755>, ZPZV<307>, ZPZV<3>>; };  // NOLINT
05512     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<1>, ZPZV<808>>; };  // NOLINT
05513     template<> struct ConwayPolynomial<811, 8> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<3>>; };  //
      NOLINT
05514     template<> struct ConwayPolynomial<811, 9> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<382>, ZPZV<200>,
      ZPZV<808>>; };  // NOLINT
05515     template<> struct ConwayPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<819>>; };  // NOLINT
05516     template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2>>; };  // NOLINT
05517     template<> struct ConwayPolynomial<821, 3> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<819>>; };  // NOLINT
05518     template<> struct ConwayPolynomial<821, 4> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<662>, ZPZV<2>>; };  // NOLINT
05519     template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<819>>; };  // NOLINT
05520     template<> struct ConwayPolynomial<821, 6> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<160>, ZPZV<130>, ZPZV<803>, ZPZV<2>>; };  // NOLINT
05521     template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<819>>; };  // NOLINT
05522     template<> struct ConwayPolynomial<821, 8> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<626>, ZPZV<556>, ZPZV<589>, ZPZV<2>>; };  //
      NOLINT
05523     template<> struct ConwayPolynomial<821, 9> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<650>, ZPZV<557>, ZPZV<819>>;
      };  // NOLINT
05524     template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<820>>; };  // NOLINT
05525     template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3>>; };  // NOLINT
05526     template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820>>; };  // NOLINT
05527     template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3>>; };  // NOLINT
05528     template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<820>>; };  // NOLINT
05529     template<> struct ConwayPolynomial<823, 6> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<822>, ZPZV<616>, ZPZV<744>, ZPZV<3>>; };  // NOLINT
05530     template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<820>>; };  // NOLINT
05531     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
05532     template<> struct ConwayPolynomial<823, 9> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<740>, ZPZV<609>, ZPZV<820>>;
      };  // NOLINT
05533     template<> struct ConwayPolynomial<827, 1> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<825>>; };  // NOLINT
05534     template<> struct ConwayPolynomial<827, 2> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2>>; };  // NOLINT
05535     template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
```

```
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825>>; };  // NOLINT
05536     template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2>>; };  // NOLINT
05537     template<> struct ConwayPolynomial<827, 5> { using ZPZ = aerobus::zpz<827>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<825>>; };  // NOLINT
05538     template<> struct ConwayPolynomial<827, 6> { using ZPZ = aerobus::zpz<827>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<685>, ZPZV<601>, ZPZV<691>, ZPZV<2>>; };  // NOLINT
05539     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<825>>; };  // NOLINT
05540     template<> struct ConwayPolynomial<827, 8> { using ZPZ = aerobus::zpz<827>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<812>, ZPZV<79>, ZPZV<32>, ZPZV<2>>; };  //
    NOLINT
05541     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>>;
    };  // NOLINT
05542     template<> struct ConwayPolynomial<829, 1> { using ZPZ = aerobus::zpz<829>; using type =
    POLYV<ZPZV<1>, ZPZV<827>>; };  // NOLINT
05543     template<> struct ConwayPolynomial<829, 2> { using ZPZ = aerobus::zpz<829>; using type =
    POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2>>; };  // NOLINT
05544     template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<827>>; };  // NOLINT
05545     template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<604>, ZPZV<2>>; };  // NOLINT
05546     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
05547     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
05548     template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<827>>; };  // NOLINT
05549     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
05550     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
05551     template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
    POLYV<ZPZV<1>, ZPZV<828>>; };  // NOLINT
05552     template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
    POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11>>; };  // NOLINT
05553     template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828>>; };  // NOLINT
05554     template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11>>; };  // NOLINT
05555     template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<828>>; };  // NOLINT
05556     template<> struct ConwayPolynomial<839, 6> { using ZPZ = aerobus::zpz<839>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<370>, ZPZV<537>, ZPZV<23>, ZPZV<11>>; };  // NOLINT
05557     template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<828>>; };  // NOLINT
05558     template<> struct ConwayPolynomial<839, 8> { using ZPZ = aerobus::zpz<839>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<553>, ZPZV<779>, ZPZV<329>, ZPZV<11>>; };  //
    NOLINT
05559     template<> struct ConwayPolynomial<839, 9> { using ZPZ = aerobus::zpz<839>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<349>, ZPZV<206>, ZPZV<828>>;
    };  // NOLINT
05560     template<> struct ConwayPolynomial<853, 1> { using ZPZ = aerobus::zpz<853>; using type =
    POLYV<ZPZV<1>, ZPZV<851>>; };  // NOLINT
05561     template<> struct ConwayPolynomial<853, 2> { using ZPZ = aerobus::zpz<853>; using type =
    POLYV<ZPZV<1>, ZPZV<852>, ZPZV<2>>; };  // NOLINT
05562     template<> struct ConwayPolynomial<853, 3> { using ZPZ = aerobus::zpz<853>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851>>; };  // NOLINT
05563     template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<2>>; };  // NOLINT
05564     template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<851>>; };  // NOLINT
05565     template<> struct ConwayPolynomial<853, 6> { using ZPZ = aerobus::zpz<853>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<276>, ZPZV<194>, ZPZV<512>, ZPZV<2>>; };  // NOLINT
05566     template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<851>>; };  // NOLINT
05567     template<> struct ConwayPolynomial<853, 8> { using ZPZ = aerobus::zpz<853>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<544>, ZPZV<846>, ZPZV<118>, ZPZV<2>>; };  //
    NOLINT
05568     template<> struct ConwayPolynomial<853, 9> { using ZPZ = aerobus::zpz<853>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<677>, ZPZV<821>,
    ZPZV<851>>; };  // NOLINT
05569     template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
    POLYV<ZPZV<1>, ZPZV<854>>; };  // NOLINT
05570     template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
    POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3>>; };  // NOLINT
05571     template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854>>; };  // NOLINT
05572     template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<528>, ZPZV<3>>; };  // NOLINT
05573     template<> struct ConwayPolynomial<857, 5> { using ZPZ = aerobus::zpz<857>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<854>>; };  // NOLINT
05574     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
```

```
05575     template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::zpz<857>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<854>>; };  // NOLINT
05576     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>>; };  //
      NOLINT
05577     template<> struct ConwayPolynomial<857, 9> { using ZPZ = aerobus::zpz<857>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<308>, ZPZV<719>, ZPZV<854>>;
      };  // NOLINT
05578     template<> struct ConwayPolynomial<859, 1> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<857>>; };  // NOLINT
05579     template<> struct ConwayPolynomial<859, 2> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2>>; };  // NOLINT
05580     template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857>>; };  // NOLINT
05581     template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2>>; };  // NOLINT
05582     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
05583     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
05584     template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<857>>; };  // NOLINT
05585     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<2>>; };  //
      NOLINT
05586     template<> struct ConwayPolynomial<859, 9> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<648>, ZPZV<845>, ZPZV<857>>;
      };  // NOLINT
05587     template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<858>>; };  // NOLINT
05588     template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5>>; };  // NOLINT
05589     template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<858>>; };  // NOLINT
05590     template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5>>; };  // NOLINT
05591     template<> struct ConwayPolynomial<863, 5> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<858>>; };  // NOLINT
05592     template<> struct ConwayPolynomial<863, 6> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<330>, ZPZV<62>, ZPZV<300>, ZPZV<5>>; };  // NOLINT
05593     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
05594     template<> struct ConwayPolynomial<863, 8> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<765>, ZPZV<576>, ZPZV<849>, ZPZV<5>>; };  //
      NOLINT
05595     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
05596     template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<875>>; };  // NOLINT
05597     template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2>>; };  // NOLINT
05598     template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875>>; };  // NOLINT
05599     template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2>>; };  // NOLINT
05600     template<> struct ConwayPolynomial<877, 5> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<875>>; };  // NOLINT
05601     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
05602     template<> struct ConwayPolynomial<877, 7> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<875>>; };  // NOLINT
05603     template<> struct ConwayPolynomial<877, 8> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<767>, ZPZV<319>, ZPZV<347>, ZPZV<2>>; };  //
      NOLINT
05604     template<> struct ConwayPolynomial<877, 9> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<770>, ZPZV<278>, ZPZV<875>>;
      };  // NOLINT
05605     template<> struct ConwayPolynomial<881, 1> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<878>>; };  // NOLINT
05606     template<> struct ConwayPolynomial<881, 2> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3>>; };  // NOLINT
05607     template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<878>>; };  // NOLINT
05608     template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<447>, ZPZV<3>>; };  // NOLINT
05609     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
05610     template<> struct ConwayPolynomial<881, 6> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<419>, ZPZV<231>, ZPZV<3>>; };  // NOLINT
05611     template<> struct ConwayPolynomial<881, 7> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<878>>; };  // NOLINT
05612     template<> struct ConwayPolynomial<881, 8> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<635>, ZPZV<490>, ZPZV<561>, ZPZV<3>>; };  //
      NOLINT
05613     template<> struct ConwayPolynomial<881, 9> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<587>, ZPZV<510>, ZPZV<878>>;
```

```
    };  // NOLINT
05614     template<> struct ConwayPolynomial<883, 1> { using ZPZ = aerobus::zpz<883>; using type =
    POLYV<ZPZV<1>, ZPZV<881>>; };  // NOLINT
05615     template<> struct ConwayPolynomial<883, 2> { using ZPZ = aerobus::zpz<883>; using type =
    POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2>>; };  // NOLINT
05616     template<> struct ConwayPolynomial<883, 3> { using ZPZ = aerobus::zpz<883>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<881>>; };  // NOLINT
05617     template<> struct ConwayPolynomial<883, 4> { using ZPZ = aerobus::zpz<883>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2>>; };  // NOLINT
05618     template<> struct ConwayPolynomial<883, 5> { using ZPZ = aerobus::zpz<883>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<881>>; };  // NOLINT
05619     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
05620     template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<881>>; };  // NOLINT
05621     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<2>>; };  //
    NOLINT
05622     template<> struct ConwayPolynomial<883, 9> { using ZPZ = aerobus::zpz<883>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<360>, ZPZV<557>, ZPZV<881>>;
    };  // NOLINT
05623     template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
    POLYV<ZPZV<1>, ZPZV<882>>; };  // NOLINT
05624     template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
    POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5>>; };  // NOLINT
05625     template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<882>>; };  // NOLINT
05626     template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5>>; };  // NOLINT
05627     template<> struct ConwayPolynomial<887, 5> { using ZPZ = aerobus::zpz<887>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<882>>; };  // NOLINT
05628     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
05629     template<> struct ConwayPolynomial<887, 7> { using ZPZ = aerobus::zpz<887>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<882>>; };  // NOLINT
05630     template<> struct ConwayPolynomial<887, 8> { using ZPZ = aerobus::zpz<887>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<781>, ZPZV<381>, ZPZV<706>, ZPZV<5>>; };  //
    NOLINT
05631     template<> struct ConwayPolynomial<887, 9> { using ZPZ = aerobus::zpz<887>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<727>, ZPZV<345>, ZPZV<882>>;
    };  // NOLINT
05632     template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
    POLYV<ZPZV<1>, ZPZV<905>>; };  // NOLINT
05633     template<> struct ConwayPolynomial<907, 2> { using ZPZ = aerobus::zpz<907>; using type =
    POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2>>; };  // NOLINT
05634     template<> struct ConwayPolynomial<907, 3> { using ZPZ = aerobus::zpz<907>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<905>>; };  // NOLINT
05635     template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2>>; };  // NOLINT
05636     template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905>>; };  // NOLINT
05637     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
05638     template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905>>; };  // NOLINT
05639     template<> struct ConwayPolynomial<907, 8> { using ZPZ = aerobus::zpz<907>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<584>, ZPZV<518>, ZPZV<811>, ZPZV<2>>; };  //
    NOLINT
05640     template<> struct ConwayPolynomial<907, 9> { using ZPZ = aerobus::zpz<907>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<783>, ZPZV<57>, ZPZV<905>>;
    };  // NOLINT
05641     template<> struct ConwayPolynomial<911, 1> { using ZPZ = aerobus::zpz<911>; using type =
    POLYV<ZPZV<1>, ZPZV<894>>; };  // NOLINT
05642     template<> struct ConwayPolynomial<911, 2> { using ZPZ = aerobus::zpz<911>; using type =
    POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17>>; };  // NOLINT
05643     template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894>>; };  // NOLINT
05644     template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17>>; };  // NOLINT
05645     template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<894>>; };  // NOLINT
05646     template<> struct ConwayPolynomial<911, 6> { using ZPZ = aerobus::zpz<911>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<172>, ZPZV<683>, ZPZV<19>, ZPZV<17>>; };  // NOLINT
05647     template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<894>>; };  // NOLINT
05648     template<> struct ConwayPolynomial<911, 8> { using ZPZ = aerobus::zpz<911>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<590>, ZPZV<168>, ZPZV<17>>; };  //
    NOLINT
05649     template<> struct ConwayPolynomial<911, 9> { using ZPZ = aerobus::zpz<911>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<679>, ZPZV<116>, ZPZV<894>>;
    };  // NOLINT
05650     template<> struct ConwayPolynomial<919, 1> { using ZPZ = aerobus::zpz<919>; using type =
    POLYV<ZPZV<1>, ZPZV<912>>; };  // NOLINT
05651     template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
    POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7>>; };  // NOLINT
05652     template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912>>; };  // NOLINT
```

```
05653     template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7>>; };  // NOLINT
05654     template<> struct ConwayPolynomial<919, 5> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<912>>; };  // NOLINT
05655     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
05656     template<> struct ConwayPolynomial<919, 7> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<912>>; };  // NOLINT
05657     template<> struct ConwayPolynomial<919, 8> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<202>, ZPZV<504>, ZPZV<7>>; };  //
      NOLINT
05658     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
05659     template<> struct ConwayPolynomial<929, 1> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<926>>; };  // NOLINT
05660     template<> struct ConwayPolynomial<929, 2> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3>>; };  // NOLINT
05661     template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<926>>; };  // NOLINT
05662     template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3>>; };  // NOLINT
05663     template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<926>>; };  // NOLINT
05664     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
05665     template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<926>>; };  // NOLINT
05666     template<> struct ConwayPolynomial<929, 8> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699>, ZPZV<292>, ZPZV<586>, ZPZV<3>>; };  //
      NOLINT
05667     template<> struct ConwayPolynomial<929, 9> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<481>, ZPZV<199>, ZPZV<926>>;
      };  // NOLINT
05668     template<> struct ConwayPolynomial<937, 1> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<932>>; };  // NOLINT
05669     template<> struct ConwayPolynomial<937, 2> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5>>; };  // NOLINT
05670     template<> struct ConwayPolynomial<937, 3> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<932>>; };  // NOLINT
05671     template<> struct ConwayPolynomial<937, 4> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<23>, ZPZV<585>, ZPZV<5>>; };  // NOLINT
05672     template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<932>>; };  // NOLINT
05673     template<> struct ConwayPolynomial<937, 6> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<794>, ZPZV<727>, ZPZV<934>, ZPZV<5>>; };  // NOLINT
05674     template<> struct ConwayPolynomial<937, 7> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<932>>; };  // NOLINT
05675     template<> struct ConwayPolynomial<937, 8> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<26>, ZPZV<53>, ZPZV<5>>; };  //
      NOLINT
05676     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
05677     template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<939>>; };  // NOLINT
05678     template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2>>; };  // NOLINT
05679     template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<939>>; };  // NOLINT
05680     template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2>>; };  // NOLINT
05681     template<> struct ConwayPolynomial<941, 5> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<939>>; };  // NOLINT
05682     template<> struct ConwayPolynomial<941, 6> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<459>, ZPZV<694>, ZPZV<538>, ZPZV<2>>; };  // NOLINT
05683     template<> struct ConwayPolynomial<941, 7> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<939>>; };  // NOLINT
05684     template<> struct ConwayPolynomial<941, 8> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<675>, ZPZV<590>, ZPZV<2>>; };  //
      NOLINT
05685     template<> struct ConwayPolynomial<941, 9> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708>, ZPZV<197>, ZPZV<939>>;
      };  // NOLINT
05686     template<> struct ConwayPolynomial<947, 1> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<945>>; };  // NOLINT
05687     template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2>>; };  // NOLINT
05688     template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<945>>; };  // NOLINT
05689     template<> struct ConwayPolynomial<947, 4> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2>>; };  // NOLINT
05690     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
05691     template<> struct ConwayPolynomial<947, 6> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<880>, ZPZV<787>, ZPZV<95>, ZPZV<2>>; };  // NOLINT
05692     template<> struct ConwayPolynomial<947, 7> { using ZPZ = aerobus::zpz<947>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<945>>; };  // NOLINT
05693     template<> struct ConwayPolynomial<947, 8> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<845>, ZPZV<597>, ZPZV<581>, ZPZV<2>>; };  //
      NOLINT
05694     template<> struct ConwayPolynomial<947, 9> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<269>, ZPZV<808>, ZPZV<945>>;
      };  // NOLINT
05695     template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<950>>; };  // NOLINT
05696     template<> struct ConwayPolynomial<953, 2> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3>>; };  // NOLINT
05697     template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950>>; };  // NOLINT
05698     template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<865>, ZPZV<3>>; };  // NOLINT
05699     template<> struct ConwayPolynomial<953, 5> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<950>>; };  // NOLINT
05700     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
05701     template<> struct ConwayPolynomial<953, 7> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<950>>; };  // NOLINT
05702     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
05703     template<> struct ConwayPolynomial<953, 9> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<819>, ZPZV<316>, ZPZV<950>>;
      };  // NOLINT
05704     template<> struct ConwayPolynomial<967, 1> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<962>>; };  // NOLINT
05705     template<> struct ConwayPolynomial<967, 2> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5>>; };  // NOLINT
05706     template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962>>; };  // NOLINT
05707     template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5>>; };  // NOLINT
05708     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
05709     template<> struct ConwayPolynomial<967, 6> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<948>, ZPZV<831>, ZPZV<5>>; };  // NOLINT
05710     template<> struct ConwayPolynomial<967, 7> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<962>>; };  // NOLINT
05711     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
05712     template<> struct ConwayPolynomial<967, 9> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<512>, ZPZV<783>, ZPZV<962>>;
      };  // NOLINT
05713     template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<965>>; };  // NOLINT
05714     template<> struct ConwayPolynomial<971, 2> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<970>, ZPZV<6>>; };  // NOLINT
05715     template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<965>>; };  // NOLINT
05716     template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<527>, ZPZV<6>>; };  // NOLINT
05717     template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<965>>; };  // NOLINT
05718     template<> struct ConwayPolynomial<971, 6> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<970>, ZPZV<729>, ZPZV<718>, ZPZV<6>>; };  // NOLINT
05719     template<> struct ConwayPolynomial<971, 7> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<965>>; };  // NOLINT
05720     template<> struct ConwayPolynomial<971, 8> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<725>, ZPZV<281>, ZPZV<206>, ZPZV<6>>; };  //
      NOLINT
05721     template<> struct ConwayPolynomial<971, 9> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<473>, ZPZV<965>>;
      };  // NOLINT
05722     template<> struct ConwayPolynomial<977, 1> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<974>>; };  // NOLINT
05723     template<> struct ConwayPolynomial<977, 2> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3>>; };  // NOLINT
05724     template<> struct ConwayPolynomial<977, 3> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<974>>; };  // NOLINT
05725     template<> struct ConwayPolynomial<977, 4> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<800>, ZPZV<3>>; };  // NOLINT
05726     template<> struct ConwayPolynomial<977, 5> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<974>>; };  // NOLINT
05727     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
05728     template<> struct ConwayPolynomial<977, 7> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<974>>; };  // NOLINT
05729     template<> struct ConwayPolynomial<977, 8> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<855>, ZPZV<807>, ZPZV<77>, ZPZV<3>>; };  //
      NOLINT
05730     template<> struct ConwayPolynomial<977, 9> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<740>, ZPZV<974>>;
      };  // NOLINT
```

```
05731     template<> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<978>>; };  // NOLINT
05732     template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5>>; };  // NOLINT
05733     template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978>>; };  // NOLINT
05734     template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5>>; };  // NOLINT
05735     template<> struct ConwayPolynomial<983, 5> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<978>>; };  // NOLINT
05736     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
05737     template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<978>>; };  // NOLINT
05738     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<5>>; };  //
      NOLINT
05739     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>>;
      };  // NOLINT
05740     template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<985>>; };  // NOLINT
05741     template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6>>; };  // NOLINT
05742     template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985>>; };  // NOLINT
05743     template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6>>; };  // NOLINT
05744     template<> struct ConwayPolynomial<991, 5> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<985>>; };  // NOLINT
05745     template<> struct ConwayPolynomial<991, 6> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<637>, ZPZV<855>, ZPZV<278>, ZPZV<6>>; };  // NOLINT
05746     template<> struct ConwayPolynomial<991, 7> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<985>>; };  // NOLINT
05747     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
05748     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<466>, ZPZV<222>, ZPZV<985>>;
      };  // NOLINT
05749     template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<990>>; };  // NOLINT
05750     template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7>>; };  // NOLINT
05751     template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990>>; };  // NOLINT
05752     template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7>>; };  // NOLINT
05753     template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<990>>; };  // NOLINT
05754     template<> struct ConwayPolynomial<997, 6> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<981>, ZPZV<58>, ZPZV<260>, ZPZV<7>>; };  // NOLINT
05755     template<> struct ConwayPolynomial<997, 7> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<990>>; };  // NOLINT
05756     template<> struct ConwayPolynomial<997, 8> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<934>, ZPZV<473>, ZPZV<241>, ZPZV<7>>; };  //
      NOLINT
05757     template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<732>, ZPZV<616>,
      ZPZV<990>>; };  // NOLINT
05758 #endif  // DO_NOT_DOCUMENT
05759 }  // namespace aerobus
05760 #endif  // AEROBUS_CONWAY_IMPORTS
05761
05762 #endif // __INC_AEROBUS__ // NOLINT
```

Chapter 10

Examples

10.1 QuotientRing

inject a 'constant' in quotient ring

inject a 'constant' in quotient ring<i32, i32::val<2>>::inject_constant_t<1>

Template Parameters

x	a 'constant' from Ring point of view

10.2 type_list

A list of types <int, double, float>

A list of types <int, double, float>

Template Parameters

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

10.3 i32::template

inject a native constant

inject a native constant

Template Parameters

x	inject_constant_2<2> -> i32::template val<2>

10.4 i32::add_t

addition operator yields v1 + v2 <i32::val<2>, i32::val<3>>

addition operator yields v1 + v2 <i32::val<2>, i32::val<3>>

Template Parameters

v1	a value in i32
v2	a value in i32

10.5 i32::sub_t

substraction operator yields v1 - v2 <i32::val<3>, i32::val<2>>

substraction operator yields v1 - v2 <i32::val<3>, i32::val<2>>

Template Parameters

v1	a value in i32
v2	a value in i32

10.6 i32::mul_t

multiplication operator yields v1 ∗ v2 <i32::val<3>, i32::val<2>>

multiplication operator yields v1 ∗ v2 <i32::val<3>, i32::val<2>>

Template Parameters

v1	a value in i32
v2	a value in i32

10.7 i32::div_t

division operator yields v1 / v2 <i32::val<7>, i32::val<2>> -> i32::val<3>

division operator yields v1 / v2 <i32::val<7>, i32::val<2>> -> i32::val<3>

Template Parameters

v1	a value in i32
v2	a value in i32

10.8 i32::gt_t

strictly greater operator (v1 > v2) yields v1 > v2 <i32::val<7>, i32::val<2>>

strictly greater operator (v1 > v2) yields v1 > v2 <i32::val<7>, i32::val<2>>

Template Parameters

v1	a value in i32
v2	a value in i32

10.9 i32::eq_t

equality operator (type) yields v1 == v2 as std::integral_constant<bool> <i32::val<2>, i32::val<2>>

equality operator (type) yields v1 == v2 as std::integral_constant<bool> <i32::val<2>, i32::val<2>>

Template Parameters

v1	a value in i32
v2	a value in i32

10.10 i32::eq_v

equality operator (boolean value)

equality operator (boolean value)

Template Parameters

v1	
v2	<i32::val<1>, i32::val<1>>

10.11 i32::gcd_t

greatest common divisor yields GCD(v1, v2) <i32::val<6>, i32::val<15>>

greatest common divisor yields GCD(v1, v2) <i32::val<6>, i32::val<15>>

Template Parameters

v1	a value in i32
v2	a value in i32

10.12 i32::pos_t

positivity operator yields v > 0 as std::true_type or std::false_type <i32::val<1

positivity operator yields v > 0 as std::true_type or std::false_type <i32::val<1

Template Parameters

v	a value in i32

10.13 i32::pos_v

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

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

Template Parameters

v	a value in i32 <i32::val<1>>

10.14 i64::template

injects constant as an i64 value

injects constant as an i64 value

Template Parameters

x	inject_constant_t<2>

10.15 i64::add_t

addition operator

addition operator

Template Parameters

v1	: an element of aerobus::i64::val
v2	: an element of aerobus::i64::val <i64::val<1>, i64::val<2>>

10.16 i64::sub_t

substraction operator

substraction operator

Template Parameters

v1	: an element of aerobus::i64::val
v2	: an element of aerobus::i64::val <i64::val<1>, i64::val<2>>

10.17 i64::mul_t

multiplication operator

multiplication operator

Template Parameters

v1	: an element of aerobus::i64::val
v2	: an element of aerobus::i64::val <i64::val<1>, i64::val<2>>

10.18 i64::div_t

division operator integer division

division operator integer division

Template Parameters

v1	: an element of aerobus::i64::val
v2	: an element of aerobus::i64::val <i64::val<1>, i64::val<2>>

10.19 i64::mod_t

modulus operator

modulus operator

Template Parameters

v1	: an element of aerobus::i64::val
v2	: an element of aerobus::i64::val <i64::val<6>, i64::val<15>>

10.20 i64::gt_t

strictly greater operator yields v1 > v2 as std::true_type or std::false_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 <i64::val<2>, i64::val<1>>

10.21 i64::lt_t

strict less operator yields v1 < v2 as std::true_type or std::false_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 <i64::val<1>, i64::val<2>>

10.22 i64::lt_v

strictly smaller operator yields v1 < v2 as boolean value

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 <i64::val<1>, i64::val<2>>

10.23 i64::eq_t

equality operator yields v1 == v2 as std::true_type or std::false_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 <i64::val<2>, i64::val<2>>

10.24 i64::eq_v

equality operator yields v1 == v2 as boolean value

equality operator yields v1 == v2 as boolean value

Template Parameters

v1	: an element of aerobus::i64::val
v2	: an element of aerobus::i64::val <i64::val<2>, i64::val<2>>

10.25 i64::gcd_t

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

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 <i64::val<6>, i64::val<15>>

10.26 i64::pos_t

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

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

Template Parameters

v1	: an element of aerobus::i64::val <i64::val<1>>

10.27 i64::pos_v

positivity yields v > 0 as boolean value

positivity yields v > 0 as boolean value

Template Parameters

v	: an element of aerobus::i64::val <i64::val<1>>

10.28 polynomial

makes the constant (native type) polynomial a_0

makes the constant (native type) polynomial a_0

Template Parameters

x	<i32>::template inject_constant_t<2>

10.29 q32::add_t

addition operator

addition operator

Template Parameters

v1	a value
v2	a value <q32::val<i32::val<1>, i32::val<2>>, q32::val<i32::val<1>, i32::val<3>>>

10.30 FractionField

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

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

Template Parameters

Ring	<i64> is q64 (rationals with 64 bits numerator and denominator)

10.31 PI_fraction::val

representation of π as a continued fraction -> 3.14...

10.32 E_fraction::val

approximation of e -> 2.718...

approximation of e -> 2.718...

Index