Aerobus

v1.2

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 '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 Structures

1.1.1 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>

1.1.2 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.

1.1.3 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
```

1.1.4 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>;
```

1.1.5 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:
```cpp
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
```

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

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

1.2 Operations

1.2.1 Fraction Field

1.2.2 Quotient

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
- 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 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

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
- 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<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 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 , int k>
  using alternate_t = typename internal::alternate< T, k >::type

  (-1)$^\wedge$ k as type in T

- template<typename T , int n, int k>
  using stirling_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 , auto p, auto n>
  using pow_t = typename internal::pow< T, p, n >::type

  p$^\wedge$ n (as 'val' type in T)

- template<typename T , template< typename, size_t index > typename coeff_at, size_t deg>
  using taylor = typename internal::make_taylor_impl< T, coeff_at, internal::make_index_sequence_reverse< deg+1 > >::type

- template<typename Integers , size_t deg>
  using exp = taylor< Integers, internal::exp_coeff, deg >

  e^x

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

  $e^x - 1$

- template<typename Integers , size_t deg>
  using lnp1 = taylor< Integers, internal::lnp1_coeff, deg >

  $\ln(1 + x)$

- template<typename Integers , size_t deg>
  using atan = taylor< Integers, internal::atan_coeff, deg >

  $\arctan(x)$

- template<typename Integers , size_t deg>
  using sin = taylor< Integers, internal::sin_coeff, deg >

  $\sin(x)$

- 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 >

  $\text{arcsinh}(x)$ *arc hyperbolic sinus*
- template<typename Integers , size_t deg>
  using atanh = taylor< Integers, internal::atanh_coeff, deg >

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

  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)$^\wedge$ 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 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>

6.1.2.3 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

6.1.2.4 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

6.1.2.5 asinh

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

$\mathrm{arcsinh}(x)$ arc hyperbolic sinus

Template Parameters

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

6.1.2.6 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

6.1.2.7 atanh

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

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

Template Parameters

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

6.1.2.8 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	

6.1.2.9 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)

6.1.2.10 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

6.1.2.11 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

6.1.2.12 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>

6.1.2.13 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

6.1.2.14 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

6.1.2.15 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	

6.1.2.16 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)

6.1.2.17 fpq64

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

polynomial with 64 bits rational coefficients

6.1.2.18 FractionField

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

6.1.2.19 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)

6.1.2.20 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

6.1.2.21 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

6.1.2.22 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

6.1.2.23 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

6.1.2.24 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

6.1.2.25 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>

6.1.2.26 pi64

using aerobus::pi64 = typedef polynomial<i64>

polynomial with 64 bits integers coefficients

6.1.2.27 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>

6.1.2.28 pow_t

```
template<typename T , auto p, auto 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	(from T::inner_type, such as int64_t)
n	(from T::inner_type, such as int64_t)

6.1.2.29 pq64

using aerobus::pq64 = typedef polynomial<q64>

polynomial with 64 bits rationals coefficients

6.1.2.30 q32

using aerobus::q32 = typedef FractionField<i32>

32 bits rationals rationals with 32 bits numerator and denominator

6.1.2.31 q64

using aerobus::q64 = typedef FractionField<i64>

64 bits rationals rationals with 64 bits numerator and denominator

6.1.2.32 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

6.1.2.33 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

6.1.2.34 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, 2, 2>
approximation of $\sqrt{2}$

6.1.2.35 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

6.1.2.36 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)

6.1.2.37   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)

6.1.2.38   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

6.1.2.39   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

6.1.2.40 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_sequenc
+ 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	

6.1.2.41 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	

6.1.2.42 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<Float↩
Type>  [inline], [constexpr]
```

nth bernoulli number as value in FloatType

Template Parameters

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

6.1.4.3 combination_v

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

computes binomial coefficients (k among n) as value

Template Parameters

T	([aerobus::i32](#) for example)
k	
n	

6.1.4.4 factorial_v

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

computes factorial(i) as value in T

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 **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** >
- struct **bernstein_helper**< **i, m, std::enable_if_t**<(**m** > **0) &&(i** > **0) &&(i**< **m)**> >
- struct **bernstein_helper**< **i, m, std::enable_if_t**<(**m** > **0) &&(i==0)**> >
- struct **bernstein_helper**< **i, m, std::enable_if_t**<(**m** > **0) &&(i==m)**> >
- struct **chebyshev_helper**
- struct **chebyshev_helper**< **1, 0** >
- struct **chebyshev_helper**< **1, 1** >
- struct **chebyshev_helper**< **2, 0** >
- struct **chebyshev_helper**< **2, 1** >
- 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 **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** >
- struct **hermite_helper**< **0, known_polynomials::hermite_kind::probabilist** >
- struct **hermite_helper**< **1, known_polynomials::hermite_kind::physicist** >
- struct **hermite_helper**< **1, known_polynomials::hermite_kind::probabilist** >
- struct **hermite_helper**< **deg, known_polynomials::hermite_kind::physicist** >
- struct **hermite_helper**< **deg, known_polynomials::hermite_kind::probabilist** >
- struct **insert_h**
- struct **is_instantiation_of**
- struct **is_instantiation_of**< **TT, TT**< **Ts...** > >
- struct **laguerre_helper**
- struct **laguerre_helper**< **0** >
- struct **laguerre_helper**< **1** >
- struct **legendre_helper**
- struct **legendre_helper**< **0** >
- struct **legendre_helper**< **1** >
- struct **lnp1_coeff**
- struct **lnp1_coeff**< **T, 0** >
- struct **make_taylor_impl**
- struct **make_taylor_impl**< **T, coeff_at, std::integer_sequence**< **size_t, ls...** > >
- struct **pop_front_h**
- struct **pow**
- struct **pow**< **T, n, p, std::enable_if_t**< **p==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 **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 **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>
  using chebyshev_T = typename internal::chebyshev_helper< 1, deg >::type

  Chebyshev polynomials of first kind.

- template<size_t deg>
  using chebyshev_U = typename internal::chebyshev_helper< 2, deg >::type

  Chebyshev polynomials of second kind.

- template<size_t deg>
  using laguerre = typename internal::laguerre_helper< deg >::type

  Laguerre polynomials.

- template<size_t deg>
  using hermite_prob = typename internal::hermite_helper< deg, hermite_kind::probabilist >::type

  Hermite polynomials - probabilist form.

- template<size_t deg>
  using hermite_phys = typename internal::hermite_helper< deg, hermite_kind::physicist >::type

  Hermite polynomials - physicist form.

- template<size_t i, size_t m>
  using bernstein = typename internal::bernstein_helper< i, m >::type

  Bernstein polynomials.

- template<size_t deg>
  using legendre = typename internal::legendre_helper< deg >::type

  Legendre polynomials.

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

Bernoulli polynomials.

See also

    See in Wikipedia

Template Parameters

deg	degree of polynomial

6.3.2.2 bernstein

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

Bernstein polynomials.

See also

    See in Wikipedia

Template Parameters

i	index of polynomial (between 0 and m)
m	degree of polynomial

6.3.2.3 chebyshev_T

```
template<size_t deg>
```

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

Chebyshev polynomials of first kind.

See also

> See in Wikipedia

Template Parameters

deg	degree of polynomial

6.3.2.4 chebyshev_U

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

Chebyshev polynomials of second kind.

See also

> See in Wikipedia

Template Parameters

deg	degree of polynomial

6.3.2.5 hermite_phys

```
template<size_t deg>
using aerobus::known_polynomials::hermite_phys = typedef typename internal::hermite_helper<deg,
hermite_kind::physicist>::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>
using aerobus::known_polynomials::hermite_prob = typedef typename internal::hermite_helper<deg,
hermite_kind::probabilist>::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>
using aerobus::known_polynomials::laguerre = typedef typename internal::laguerre_helper<deg>↩
::type
```

Laguerre polynomials.

See also

See in Wikipedia

Template Parameters

deg	degree of polynomial

6.3.2.8 legendre

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

Legendre polynomials.

See also

See in Wikipedia

Template Parameters

deg	degree of polynomial

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

```cpp
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

```cpp
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**

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

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>()
    represented 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]
```

represented 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::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

8.8.1 Detailed Description

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

8.8.2 Member Typedef Documentation

8.8.2.1 add_t

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

8.8.2.2 div_t

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

8.8.2.3 eq_t

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

8.8.2.4 gcd_t

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

8.8.2.5 gt_t

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

8.8.2.6 inject_constant_t

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

8.8.2.7 inject_ring_t

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

8.8.2.8 inner_type

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

8.8.2.9 lt_t

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

8.8.2.10 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

8.8.2.11 mul_t

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

8.8.2.12 one

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

constant one

8.8.2.13 pos_t

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

8.8.2.14 sub_t

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

8.8.2.15 zero

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

constant zero

8.8.3 Member Data Documentation

8.8.3.1 eq_v

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

8.8.3.2 is_euclidean_domain

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

integers are an euclidean domain

8.8.3.3 is_field

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

integers are not a field

8.8.3.4 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

8.9 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 for actual 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

8.9.1 Detailed Description

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

8.9.2 Member Typedef Documentation

8.9.2.1 add_t

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

8.9.2.2 div_t

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

8.9.2.3 eq_t

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

8.9.2.4 gcd_t

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

8.9.2.5 gt_t

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

8.9.2.6 inject_constant_t

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

8.9.2.7 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

8.9.2.8 inner_type

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

type for actual values

8.9.2.9 lt_t

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

8.9.2.10 mod_t

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

8.9.2.11 mul_t

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

8.9.2.12 one

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

constant one

8.9.2.13 pos_t

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

8.9.2.14 sub_t

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

8.9.2.15 zero

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

constant zero

8.9.3 Member Data Documentation

8.9.3.1 eq_v

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

8.9.3.2 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

8.9.3.3 is_euclidean_domain

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

integers are an euclidean domain

8.9.3.4 is_field

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

integers are not a field

8.9.3.5 lt_v

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

8.9.3.6 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

8.10 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

8.10.1 Detailed Description

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

checks if n is prime

Template Parameters

n	

8.10.2 Member Data Documentation

8.10.2.1 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

8.11 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

8.11.1 Detailed Description

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

polynomial with coefficients in Ring Ring must be an integral domain

8.11.2 Member Typedef Documentation

8.11.2.1 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	

8.11.2.2 derive_t

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

derivation operator

Template Parameters

v	

8.11.2.3 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	

8.11.2.4 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	

8.11.2.5 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	

8.11.2.6 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	

8.11.2.7 inject_constant_t

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

8.11.2.8 inject_ring_t

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

8.11.2.9 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	

8.11.2.10 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	

8.11.2.11 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	

8.11.2.12 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	

8.11.2.13 one

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

constant one

8.11.2.14 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	

8.11.2.15 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	

8.11.2.16 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	

8.11.2.17 X

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

generator

8.11.2.18 zero

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

constant zero

8.11.3 Member Data Documentation

8.11.3.1 is_euclidean_domain

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

8.11.3.2 is_field

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

8.11.3.3 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

8.12 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

8.12.1 Detailed Description

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

removes types from head of the list

8.12.2 Member Typedef Documentation

8.12.2.1 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

8.12.2.2 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

8.13 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

8.13.1 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>

8.13.2 Member Typedef Documentation

8.13.2.1 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

8.13.2.2 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

8.13.2.3 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

8.13.2.4 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>
>
```

8.13.2.5 inject_ring_t

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

8.13.2.6 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

8.13.2.7 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

8.13.2.8 one

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

one

8.13.2.9 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

8.13.2.10 zero

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

zero value

8.13.3 Member Data Documentation

8.13.3.1 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

8.13.3.2 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

8.13.3.3 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

8.14 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

8.14.1 Detailed Description

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

splits list at index

Template Parameters

index	

8.14.2 Member Typedef Documentation

8.14.2.1 head

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


8.14.2.2 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

8.15 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

8.15.1 Detailed Description

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

Empty pure template struct to handle type list.

8.15.2 Member Typedef Documentation

8.15.2.1 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	

8.15.2.2 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	

8.15.2.3 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	

8.15.2.4 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	

8.15.2.5 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	

8.15.2.6 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	

8.15.3 Member Data Documentation

8.15.3.1 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

8.16 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

8.16.1 Detailed Description

specialization for empty type list

8.16.2 Member Typedef Documentation

8.16.2.1 concat

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

8.16.2.2 insert

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

8.16.2.3 push_back

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

8.16.2.4 push_front

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

8.16.3 Member Data Documentation

8.16.3.1 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

8.17 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 valueType get ()

    cast x into valueType
- static std::string to_string ()

    string representation of value
- template<typename valueRing >
  static constexpr valueRing eval (const valueRing &v)

    cast x into valueRing

Static Public Attributes

- static constexpr int32_t v = x

    actual value stored in val type

8.17.1 Detailed Description

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

values in i32, again represented as types

Template Parameters

x	an actual integer

8.17.2 Member Typedef Documentation

8.17.2.1 enclosing_type

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

Enclosing ring type.

8.17.2.2 is_zero_t

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

is value zero

8.17.3 Member Function Documentation

8.17.3.1 eval()

```
template<int32_t x>
template<typename valueRing >
static constexpr valueRing aerobus::i32::val< x >::eval (
          const valueRing & v )  [inline], [static], [constexpr]
```

cast x into valueRing

Template Parameters

valueRing	double for example

8.17.3.2 get()

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

cast x into valueType

Template Parameters

valueType	double for example

8.17.3.3 to_string()

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

string representation of value

8.17.4 Member Data Documentation

8.17.4.1 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

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

values in i64

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

Public Types

- 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 valueType get ()
  cast value in valueType
- static std::string to_string ()
  string representation
- template<typename valueRing >
  static constexpr valueRing eval (const valueRing &v)
  cast value in valueRing

Static Public Attributes

- static constexpr int64_t v = x
  actual value

8.18.1 Detailed Description

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

values in i64

Template Parameters

x	an actual integer

8.18.2 Member Typedef Documentation

8.18.2.1 enclosing_type

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

enclosing ring type

8.18.2.2 is_zero_t

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

is value zero

8.18.3 Member Function Documentation

8.18.3.1 eval()

```
template<int64_t x>
template<typename valueRing >
static constexpr valueRing aerobus::i64::val< x >::eval (
            const valueRing & v )  [inline], [static], [constexpr]
```

cast value in valueRing

Template Parameters

$valueRing$	(double for example)

8.18.3.2 get()

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

cast value in valueType

Template Parameters

$valueType$	(double for example)

8.18.3.3 to_string()

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

string representation

8.18.4 Member Data Documentation

8.18.4.1 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

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

values (seen as types) in polynomial ring

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

Public Types

- 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 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

8.19.1 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

8.19.2 Member Typedef Documentation

8.19.2.1 aN

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

heavy weight coefficient (non zero)

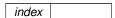### 8.19.2.2 coeff_at_t

```
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	

8.19.2.3 enclosing_type

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

enclosing ring type

8.19.2.4 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

8.19.2.5 strip

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

remove largest coefficient

8.19.3 Member Function Documentation

8.19.3.1 eval()

template<typename Ring >
template<typename coeffN , typename... coeffs>
template<typename valueRing >
static constexpr 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

$P(x)$

8.19.3.2 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$^\wedge$n + ... + a_1 X + a_0

8.19.4 Member Data Documentation

8.19.4.1 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

8.19.4.2 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

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

projection values in the quotient ring

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

Public Types

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

8.20.1 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'

8.20.2 Member Typedef Documentation

8.20.2.1 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

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

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

Public Types

- using enclosing_type = zpz< p >
  enclosing ring type
- using is_zero_t = std::bool_constant< x% p==0 >

Static Public Member Functions

- template<typename valueType >
  static constexpr valueType get ()
- static std::string to_string ()
- template<typename valueRing >
  static constexpr valueRing eval (const valueRing &v)

Static Public Attributes

- static constexpr int32_t v = x % p
  actual value

8.21.1 Member Typedef Documentation

8.21.1.1 enclosing_type

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

enclosing ring type

8.21.1.2 is_zero_t

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

8.21.2 Member Function Documentation

8.21.2.1 eval()

```
template<int32_t p>
template<int32_t x>
template<typename valueRing >
static constexpr valueRing aerobus::zpz< p >::val< x >::eval (
            const valueRing & v )  [inline], [static], [constexpr]
```

8.21.2.2 get()

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

8.21.2.3 to_string()

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

8.21.3 Member Data Documentation

8.21.3.1 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

8.22 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 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 ()
- template<typename valueRing >
  static constexpr valueRing eval (const valueRing &x)

Static Public Attributes

- static constexpr size_t degree = 0

  degree
- static constexpr bool is_zero_v = is_zero_t::value

8.22.1 Detailed Description

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

specialization for constants

Template Parameters

coeffN	

8.22.2 Member Typedef Documentation

8.22.2.1 aN

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

8.22.2.2 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$>\hookleftarrow$
::type

8.22.2.3 enclosing_type

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

enclosing ring type

8.22.2.4 is_zero_t

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

8.22.2.5 strip

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

8.22.3 Member Function Documentation

8.22.3.1 eval()

template$<$typename Ring $>$
template$<$typename coeffN $>$
template$<$typename valueRing $>$
static constexpr valueRing aerobus::polynomial$<$ Ring $>$::val$<$ coeffN $>$::eval (
            const valueRing & x) [inline], [static], [constexpr]

8.22.3.2 to_string()

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

8.22.4 Member Data Documentation

8.22.4.1 degree

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

degree

8.22.4.2 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

8.23 aerobus::zpz< p > Struct Template Reference

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

Classes

- struct val

Public Types

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

- template$<$typename v1 , typename v2 $>$
  using mod_t = typename remainder$<$ v1, v2 $>$::type
    modulo operator
- template$<$typename v1 , typename v2 $>$
  using gt_t = typename gt$<$ v1, v2 $>$::type
    strictly greater operator (type)
- template$<$typename v1 , typename v2 $>$
  using lt_t = typename lt$<$ v1, v2 $>$::type
    strictly smaller operator (type)
- template$<$typename v1 , typename v2 $>$
  using eq_t = typename eq$<$ v1, v2 $>$::type
    equality operator (type)
- template$<$typename v1 , typename v2 $>$
  using gcd_t = gcd_t$<$ i32, v1, v2 $>$
    greatest common divisor
- template$<$typename v1 $>$
  using pos_t = typename pos$<$ v1 $>$::type
    positivity operator (type)

Static Public Attributes

- static constexpr bool is_field = is_prime$<$p$>$::value
- static constexpr bool is_euclidean_domain = 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)

8.23.1 Detailed Description

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

congruence classes of integers for a modulus if p is prime, zpz is a field, otherwise an integral domain with all related operations

8.23.2 Member Typedef Documentation

8.23.2.1 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

8.23.2.2 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

8.23.2.3 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

8.23.2.4 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

8.23.2.5 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

8.23.2.6 inject_constant_t

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

8.23.2.7 inner_type

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

8.23.2.8 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

8.23.2.9 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

8.23.2.10 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

8.23.2.11 one

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

8.23.2.12 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

8.23.2.13 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

8.23.2.14 zero

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

8.23.3 Member Data Documentation

8.23.3.1 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

8.23.3.2 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

8.23.3.3 is_euclidean_domain

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

8.23.3.4 is_field

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

8.23.3.5   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

8.23.3.6   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 main_page.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
00018 #ifdef _MSC_VER
00019 #define ALIGNED(x) __declspec(align(x))
00020 #define INLINED __forceinline
00021 #else
00022 #define ALIGNED(x) __attribute__((aligned(x)))
00023 #define INLINED __attribute__((always_inline)) inline
00024 #endif
00025
00027
00029
```

```
00031
00032 // aligned allocation
00033 namespace aerobus {
00040     template<typename T>
00041     T* aligned_malloc(size_t count, size_t alignment) {
00042         #ifdef _MSC_VER
00043         return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
00044         #else
00045         return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
00046         #endif
00047     }
00048 }  // namespace aerobus
00049
00050 // concepts
00051 namespace aerobus {
00053     template <typename R>
00054     concept IsRing = requires {
00055         typename R::one;
00056         typename R::zero;
00057         typename R::template add_t<typename R::one, typename R::one>;
00058         typename R::template sub_t<typename R::one, typename R::one>;
00059         typename R::template mul_t<typename R::one, typename R::one>;
00060     };
00061
00063     template <typename R>
00064     concept IsEuclideanDomain = IsRing<R> && requires {
00065         typename R::template div_t<typename R::one, typename R::one>;
00066         typename R::template mod_t<typename R::one, typename R::one>;
00067         typename R::template gcd_t<typename R::one, typename R::one>;
00068         typename R::template eq_t<typename R::one, typename R::one>;
00069         typename R::template pos_t<typename R::one>;
00070
00071         R::template pos_v<typename R::one> == true;
00072         // typename R::template gt_t<typename R::one, typename R::zero>;
00073         R::is_euclidean_domain == true;
00074     };
00075
00077     template<typename R>
00078     concept IsField = IsEuclideanDomain<R> && requires {
00079         R::is_field == true;
00080     };
00081 }  // namespace aerobus
00082
00083 // utilities
00084 namespace aerobus {
00085     namespace internal {
00086         template<template<typename...> typename TT, typename T>
00087         struct is_instantiation_of : std::false_type { };
00088
00089         template<template<typename...> typename TT, typename... Ts>
00090         struct is_instantiation_of<TT, TT<Ts...>> : std::true_type { };
00091
00092         template<template<typename...> typename TT, typename T>
00093         inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00094
00095         template <int64_t i, typename T, typename... Ts>
00096         struct type_at {
00097             static_assert(i < sizeof...(Ts) + 1, "index out of range");
00098             using type = typename type_at<i - 1, Ts...>::type;
00099         };
00100
00101         template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00102             using type = T;
00103         };
00104
00105         template <size_t i, typename... Ts>
00106         using type_at_t = typename type_at<i, Ts...>::type;
00107
00108
00109         template<size_t n, size_t i, typename E = void>
00110         struct _is_prime {};
00111
00112         template<size_t i>
00113         struct _is_prime<0, i> {
00114             static constexpr bool value = false;
00115         };
00116
00117         template<size_t i>
00118         struct _is_prime<1, i> {
00119             static constexpr bool value = false;
00120         };
00121
00122         template<size_t i>
00123         struct _is_prime<2, i> {
00124             static constexpr bool value = true;
00125         };
00126
```

```
00127          template<size_t i>
00128          struct _is_prime<3, i> {
00129               static constexpr bool value = true;
00130          };
00131
00132          template<size_t i>
00133          struct _is_prime<5, i> {
00134               static constexpr bool value = true;
00135          };
00136
00137          template<size_t i>
00138          struct _is_prime<7, i> {
00139               static constexpr bool value = true;
00140          };
00141
00142          template<size_t n, size_t i>
00143          struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)» {
00144               static constexpr bool value = false;
00145          };
00146
00147          template<size_t n, size_t i>
00148          struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)» {
00149               static constexpr bool value = false;
00150          };
00151
00152          template<size_t n, size_t i>
00153          struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)» {
00154               static constexpr bool value = true;
00155          };
00156
00157          template<size_t n, size_t i>
00158          struct _is_prime<n, i, std::enable_if_t<(
00159               n % i == 0 &&
00160               n >= 9 &&
00161               n % 3 != 0 &&
00162               n % 2 != 0 &&
00163               i * i > n)» {
00164               static constexpr bool value = true;
00165          };
00166
00167          template<size_t n, size_t i>
00168          struct _is_prime<n, i, std::enable_if_t<(
00169               n % (i+2) == 0 &&
00170               n >= 9 &&
00171               n % 3 != 0 &&
00172               n % 2 != 0 &&
00173               i * i <= n)» {
00174               static constexpr bool value = true;
00175          };
00176
00177          template<size_t n, size_t i>
00178          struct _is_prime<n, i, std::enable_if_t<(
00179                    n % (i+2) != 0 &&
00180                    n % i != 0 &&
00181                    n >= 9 &&
00182                    n % 3 != 0 &&
00183                    n % 2 != 0 &&
00184                    (i * i <= n))» {
00185               static constexpr bool value = _is_prime<n, i+6>::value;
00186          };
00187
00188     }  // namespace internal
00189
00192     template<size_t n>
00193     struct is_prime {
00195          static constexpr bool value = internal::_is_prime<n, 5>::value;
00196     };
00197
00201     template<size_t n>
00202     static constexpr bool is_prime_v = is_prime<n>::value;
00203
00204     // gcd
00205     namespace internal {
00206          template <std::size_t... Is>
00207          constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
00208               -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
00209
00210          template <std::size_t N>
00211          using make_index_sequence_reverse
00212               = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00213
00219          template<typename Ring, typename E = void>
00220          struct gcd;
00221
00222          template<typename Ring>
00223          struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {
00224               template<typename A, typename B, typename E = void>
```

```
00225                  struct gcd_helper {};
00226
00227                  // B = 0, A > 0
00228                  template<typename A, typename B>
00229                  struct gcd_helper<A, B, std::enable_if_t<
00230                      ((B::is_zero_t::value) &&
00231                          (Ring::template gt_t<A, typename Ring::zero>::value))» {
00232                      using type = A;
00233                  };
00234
00235                  // B = 0, A < 0
00236                  template<typename A, typename B>
00237                  struct gcd_helper<A, B, std::enable_if_t<
00238                      ((B::is_zero_t::value) &&
00239                          !(Ring::template gt_t<A, typename Ring::zero>::value))» {
00240                      using type = typename Ring::template sub_t<typename Ring::zero, A>;
00241                  };
00242
00243                  // B != 0
00244                  template<typename A, typename B>
00245                  struct gcd_helper<A, B, std::enable_if_t<
00246                      (!B::is_zero_t::value)
00247                      » {
00248                  private: // NOLINT
00249                      // A / B
00250                      using k = typename Ring::template div_t<A, B>;
00251                      // A - (A/B)*B = A % B
00252                      using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
00253
00254                  public:
00255                      using type = typename gcd_helper<B, m>::type;
00256                  };
00257
00258                  template<typename A, typename B>
00259                  using type = typename gcd_helper<A, B>::type;
00260              };
00261          }  // namespace internal
00262
00263      // vadd and vmul
00264      namespace internal {
00265          template<typename... vals>
00266          struct vmul {};
00267
00268          template<typename v1, typename... vals>
00269          struct vmul<v1, vals...> {
00270              using type = typename v1::enclosing_type::template mul_t<v1, typename
        vmul<vals...>::type>;
00271          };
00272
00273          template<typename v1>
00274          struct vmul<v1> {
00275              using type = v1;
00276          };
00277
00278          template<typename... vals>
00279          struct vadd {};
00280
00281          template<typename v1, typename... vals>
00282          struct vadd<v1, vals...> {
00283              using type = typename v1::enclosing_type::template add_t<v1, typename
        vadd<vals...>::type>;
00284          };
00285
00286          template<typename v1>
00287          struct vadd<v1> {
00288              using type = v1;
00289          };
00290      }  // namespace internal
00291
00294      template<typename T, typename A, typename B>
00295      using gcd_t = typename internal::gcd<T>::template type<A, B>;
00296
00300      template<typename... vals>
00301      using vadd_t = typename internal::vadd<vals...>::type;
00302
00306      template<typename... vals>
00307      using vmul_t = typename internal::vmul<vals...>::type;
00308
00312      template<typename val>
00313      requires IsEuclideanDomain<typename val::enclosing_type>
00314      using abs_t = std::conditional_t<
00315                      val::enclosing_type::template pos_v<val>,
00316                      val, typename val::enclosing_type::template sub_t<typename
        val::enclosing_type::zero, val»;
00317 }  // namespace aerobus
00318
00319 namespace aerobus {
```

```
00324      template<typename Ring, typename X>
00325      requires IsRing<Ring>
00326      struct Quotient {
00329          template <typename V>
00330          struct val {
00331           public:
00332              using type = abs_t<typename Ring::template mod_t<V, X>>;
00333          };
00334
00336          using zero = val<typename Ring::zero>;
00337
00339          using one = val<typename Ring::one>;
00340
00344          template<typename v1, typename v2>
00345          using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00346
00350          template<typename v1, typename v2>
00351          using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00352
00356          template<typename v1, typename v2>
00357          using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00358
00362          template<typename v1, typename v2>
00363          using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00364
00368          template<typename v1, typename v2>
00369          using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00370
00374          template<typename v1, typename v2>
00375          static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00376
00380          template<typename v1>
00381          using pos_t = std::true_type;
00382
00386          template<typename v>
00387          static constexpr bool pos_v = pos_t<v>::value;
00388
00390          static constexpr bool is_euclidean_domain = true;
00391
00397          template<auto x>
00398          using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00399
00405          template<typename v>
00406          using inject_ring_t = val<v>;
00407      };
00408 }  // namespace aerobus
00409
00410 // type_list
00411 namespace aerobus {
00413      template <typename... Ts>
00414      struct type_list;
00415
00416      namespace internal {
00417          template <typename T, typename... Us>
00418          struct pop_front_h {
00419              using tail = type_list<Us...>;
00420              using head = T;
00421          };
00422
00423          template <size_t index, typename L1, typename L2>
00424          struct split_h {
00425           private:
00426              static_assert(index <= L2::length, "index ouf of bounds");
00427              using a = typename L2::pop_front::type;
00428              using b = typename L2::pop_front::tail;
00429              using c = typename L1::template push_back<a>;
00430
00431           public:
00432              using head = typename split_h<index - 1, c, b>::head;
00433              using tail = typename split_h<index - 1, c, b>::tail;
00434          };
00435
00436          template <typename L1, typename L2>
00437          struct split_h<0, L1, L2> {
00438              using head = L1;
00439              using tail = L2;
00440          };
00441
00442          template <size_t index, typename L, typename T>
00443          struct insert_h {
00444              static_assert(index <= L::length, "index ouf of bounds");
00445              using s = typename L::template split<index>;
00446              using left = typename s::head;
00447              using right = typename s::tail;
00448              using ll = typename left::template push_back<T>;
00449              using type = typename ll::template concat<right>;
00450          };
```

```
00451
00452          template <size_t index, typename L>
00453          struct remove_h {
00454              using s = typename L::template split<index>;
00455              using left = typename s::head;
00456              using right = typename s::tail;
00457              using rr = typename right::pop_front::tail;
00458              using type = typename left::template concat<rr>;
00459          };
00460      }  // namespace internal
00461
00465      template <typename... Ts>
00466      struct type_list {
00467       private:
00468          template <typename T>
00469          struct concat_h;
00470
00471          template <typename... Us>
00472          struct concat_h<type_list<Us...» {
00473              using type = type_list<Ts..., Us...>;
00474          };
00475
00476       public:
00477          static constexpr size_t length = sizeof...(Ts);
00478
00479
00482          template <typename T>
00483          using push_front = type_list<T, Ts...>;
00484
00487          template <size_t index>
00488          using at = internal::type_at_t<index, Ts...>;
00489
00491          struct pop_front {
00493              using type = typename internal::pop_front_h<Ts...>::head;
00495              using tail = typename internal::pop_front_h<Ts...>::tail;
00496          };
00497
00500          template <typename T>
00501          using push_back = type_list<Ts..., T>;
00502
00505          template <typename U>
00506          using concat = typename concat_h<U>::type;
00507
00510          template <size_t index>
00511          struct split {
00512           private:
00513              using inner = internal::split_h<index, type_list<>, type_list<Ts...»;
00514
00515           public:
00516              using head = typename inner::head;
00517              using tail = typename inner::tail;
00518          };
00519
00523          template <typename T, size_t index>
00524          using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00525
00528          template <size_t index>
00529          using remove = typename internal::remove_h<index, type_list<Ts...»::type;
00530      };
00531
00533      template <>
00534      struct type_list<> {
00535          static constexpr size_t length = 0;
00536
00537          template <typename T>
00538          using push_front = type_list<T>;
00539
00540          template <typename T>
00541          using push_back = type_list<T>;
00542
00543          template <typename U>
00544          using concat = U;
00545
00546          // TODO(jewave): assert index == 0
00547          template <typename T, size_t index>
00548          using insert = type_list<T>;
00549      };
00550 }  // namespace aerobus
00551
00552 // i32
00553 namespace aerobus {
00555      struct i32 {
00556          using inner_type = int32_t;
00559          template<int32_t x>
00560          struct val {
00562              using enclosing_type = i32;
00564              static constexpr int32_t v = x;
00565
```

```
00568                 template<typename valueType>
00569                 static constexpr valueType get() { return static_cast<valueType>(x); }
00570
00572                 using is_zero_t = std::bool_constant<x == 0>;
00573
00575                 static std::string to_string() {
00576                     return std::to_string(x);
00577                 }
00578
00581                 template<typename valueRing>
00582                 static constexpr valueRing eval(const valueRing& v) {
00583                     return static_cast<valueRing>(x);
00584                 }
00585             };
00586
00588             using zero = val<0>;
00590             using one = val<1>;
00592             static constexpr bool is_field = false;
00594             static constexpr bool is_euclidean_domain = true;
00598             template<auto x>
00599             using inject_constant_t = val<static_cast<int32_t>(x)>;
00600
00601             template<typename v>
00602             using inject_ring_t = v;
00603
00604         private:
00605             template<typename v1, typename v2>
00606             struct add {
00607                 using type = val<v1::v + v2::v>;
00608             };
00609
00610             template<typename v1, typename v2>
00611             struct sub {
00612                 using type = val<v1::v - v2::v>;
00613             };
00614
00615             template<typename v1, typename v2>
00616             struct mul {
00617                 using type = val<v1::v* v2::v>;
00618             };
00619
00620             template<typename v1, typename v2>
00621             struct div {
00622                 using type = val<v1::v / v2::v>;
00623             };
00624
00625             template<typename v1, typename v2>
00626             struct remainder {
00627                 using type = val<v1::v % v2::v>;
00628             };
00629
00630             template<typename v1, typename v2>
00631             struct gt {
00632                 using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00633             };
00634
00635             template<typename v1, typename v2>
00636             struct lt {
00637                 using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00638             };
00639
00640             template<typename v1, typename v2>
00641             struct eq {
00642                 using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00643             };
00644
00645             template<typename v1>
00646             struct pos {
00647                 using type = std::bool_constant<(v1::v > 0)>;
00648             };
00649
00650         public:
00656             template<typename v1, typename v2>
00657             using add_t = typename add<v1, v2>::type;
00658
00664             template<typename v1, typename v2>
00665             using sub_t = typename sub<v1, v2>::type;
00666
00672             template<typename v1, typename v2>
00673             using mul_t = typename mul<v1, v2>::type;
00674
00680             template<typename v1, typename v2>
00681             using div_t = typename div<v1, v2>::type;
00682
00688             template<typename v1, typename v2>
00689             using mod_t = typename remainder<v1, v2>::type;
00690
```

```
00696          template<typename v1, typename v2>
00697          using gt_t = typename gt<v1, v2>::type;
00698
00704          template<typename v1, typename v2>
00705          using lt_t = typename lt<v1, v2>::type;
00706
00712          template<typename v1, typename v2>
00713          using eq_t = typename eq<v1, v2>::type;
00714
00719          template<typename v1, typename v2>
00720          static constexpr bool eq_v = eq_t<v1, v2>::value;
00721
00727          template<typename v1, typename v2>
00728          using gcd_t = gcd_t<i32, v1, v2>;
00729
00734          template<typename v>
00735          using pos_t = typename pos<v>::type;
00736
00741          template<typename v>
00742          static constexpr bool pos_v = pos_t<v>::value;
00743      };
00744 }  // namespace aerobus
00745
00746 // i64
00747 namespace aerobus {
00749      struct i64 {
00751          using inner_type = int64_t;
00754          template<int64_t x>
00755          struct val {
00757              using enclosing_type = i64;
00759              static constexpr int64_t v = x;
00760
00763              template<typename valueType>
00764              static constexpr valueType get() { return static_cast<valueType>(x); }
00765
00767              using is_zero_t = std::bool_constant<x == 0>;
00768
00770              static std::string to_string() {
00771                  return std::to_string(x);
00772              }
00773
00776              template<typename valueRing>
00777              static constexpr valueRing eval(const valueRing& v) {
00778                  return static_cast<valueRing>(x);
00779              }
00780          };
00781
00785          template<auto x>
00786          using inject_constant_t = val<static_cast<int64_t>(x)>;
00787
00792          template<typename v>
00793          using inject_ring_t = v;
00794
00796          using zero = val<0>;
00798          using one = val<1>;
00800          static constexpr bool is_field = false;
00802          static constexpr bool is_euclidean_domain = true;
00803
00804      private:
00805          template<typename v1, typename v2>
00806          struct add {
00807              using type = val<v1::v + v2::v>;
00808          };
00809
00810          template<typename v1, typename v2>
00811          struct sub {
00812              using type = val<v1::v - v2::v>;
00813          };
00814
00815          template<typename v1, typename v2>
00816          struct mul {
00817              using type = val<v1::v* v2::v>;
00818          };
00819
00820          template<typename v1, typename v2>
00821          struct div {
00822              using type = val<v1::v / v2::v>;
00823          };
00824
00825          template<typename v1, typename v2>
00826          struct remainder {
00827              using type = val<v1::v% v2::v>;
00828          };
00829
00830          template<typename v1, typename v2>
00831          struct gt {
00832              using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
```

```
00833            };
00834
00835            template<typename v1, typename v2>
00836            struct lt {
00837                using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00838            };
00839
00840            template<typename v1, typename v2>
00841            struct eq {
00842                using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00843            };
00844
00845            template<typename v>
00846            struct pos {
00847                using type = std::bool_constant<(v::v > 0)>;
00848            };
00849
00850        public:
00855            template<typename v1, typename v2>
00856            using add_t = typename add<v1, v2>::type;
00857
00862            template<typename v1, typename v2>
00863            using sub_t = typename sub<v1, v2>::type;
00864
00869            template<typename v1, typename v2>
00870            using mul_t = typename mul<v1, v2>::type;
00871
00877            template<typename v1, typename v2>
00878            using div_t = typename div<v1, v2>::type;
00879
00884            template<typename v1, typename v2>
00885            using mod_t = typename remainder<v1, v2>::type;
00886
00892            template<typename v1, typename v2>
00893            using gt_t = typename gt<v1, v2>::type;
00894
00899            template<typename v1, typename v2>
00900            static constexpr bool gt_v = gt_t<v1, v2>::value;
00901
00907            template<typename v1, typename v2>
00908            using lt_t = typename lt<v1, v2>::type;
00909
00915            template<typename v1, typename v2>
00916            static constexpr bool lt_v = lt_t<v1, v2>::value;
00917
00923            template<typename v1, typename v2>
00924            using eq_t = typename eq<v1, v2>::type;
00925
00931            template<typename v1, typename v2>
00932            static constexpr bool eq_v = eq_t<v1, v2>::value;
00933
00939            template<typename v1, typename v2>
00940            using gcd_t = gcd_t<i64, v1, v2>;
00941
00946            template<typename v>
00947            using pos_t = typename pos<v>::type;
00948
00953            template<typename v>
00954            static constexpr bool pos_v = pos_t<v>::value;
00955        };
00956 }  // namespace aerobus
00957
00958 // z/pz
00959 namespace aerobus {
00964     template<int32_t p>
00965     struct zpz {
00966         using inner_type = int32_t;
00967         template<int32_t x>
00968         struct val {
00970             using enclosing_type = zpz<p>;
00972             static constexpr int32_t v = x % p;
00973
00974             template<typename valueType>
00975            static constexpr valueType get() { return static_cast<valueType>(x % p); }
00976
00977             using is_zero_t = std::bool_constant<x% p == 0>;
00978             static std::string to_string() {
00979                 return std::to_string(x % p);
00980             }
00981
00982             template<typename valueRing>
00983             static constexpr valueRing eval(const valueRing& v) {
00984                 return static_cast<valueRing>(x % p);
00985             }
00986         };
00987
00988         template<auto x>
```

```
00989            using inject_constant_t = val<static_cast<int32_t>(x)>;
00990
00991            using zero = val<0>;
00992            using one = val<1>;
00993            static constexpr bool is_field = is_prime<p>::value;
00994            static constexpr bool is_euclidean_domain = true;
00995
00996        private:
00997            template<typename v1, typename v2>
00998            struct add {
00999                using type = val<(v1::v + v2::v) % p>;
01000            };
01001
01002            template<typename v1, typename v2>
01003            struct sub {
01004                using type = val<(v1::v - v2::v) % p>;
01005            };
01006
01007            template<typename v1, typename v2>
01008            struct mul {
01009                using type = val<(v1::v* v2::v) % p>;
01010            };
01011
01012            template<typename v1, typename v2>
01013            struct div {
01014                using type = val<(v1::v% p) / (v2::v % p)>;
01015            };
01016
01017            template<typename v1, typename v2>
01018            struct remainder {
01019                using type = val<(v1::v% v2::v) % p>;
01020            };
01021
01022            template<typename v1, typename v2>
01023            struct gt {
01024                using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01025            };
01026
01027            template<typename v1, typename v2>
01028            struct lt {
01029                using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01030            };
01031
01032            template<typename v1, typename v2>
01033            struct eq {
01034                using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01035            };
01036
01037            template<typename v1>
01038            struct pos {
01039                using type = std::bool_constant<(v1::v > 0)>;
01040            };
01041
01042        public:
01046            template<typename v1, typename v2>
01047            using add_t = typename add<v1, v2>::type;
01048
01052            template<typename v1, typename v2>
01053            using sub_t = typename sub<v1, v2>::type;
01054
01058            template<typename v1, typename v2>
01059            using mul_t = typename mul<v1, v2>::type;
01060
01064            template<typename v1, typename v2>
01065            using div_t = typename div<v1, v2>::type;
01066
01070            template<typename v1, typename v2>
01071            using mod_t = typename remainder<v1, v2>::type;
01072
01076            template<typename v1, typename v2>
01077            using gt_t = typename gt<v1, v2>::type;
01078
01082            template<typename v1, typename v2>
01083            static constexpr bool gt_v = gt_t<v1, v2>::value;
01084
01088            template<typename v1, typename v2>
01089            using lt_t = typename lt<v1, v2>::type;
01090
01094            template<typename v1, typename v2>
01095            static constexpr bool lt_v = lt_t<v1, v2>::value;
01096
01100            template<typename v1, typename v2>
01101            using eq_t = typename eq<v1, v2>::type;
01102
01106            template<typename v1, typename v2>
01107            static constexpr bool eq_v = eq_t<v1, v2>::value;
01108
```

```
01112          template<typename v1, typename v2>
01113          using gcd_t = gcd_t<i32, v1, v2>;
01114
01117          template<typename v1>
01118          using pos_t = typename pos<v1>::type;
01119
01122          template<typename v>
01123          static constexpr bool pos_v = pos_t<v>::value;
01124      };
01125 }  // namespace aerobus
01126
01127 // polynomial
01128 namespace aerobus {
01129     // coeffN x^N + ...
01134     template<typename Ring>
01135     requires IsEuclideanDomain<Ring>
01136     struct polynomial {
01137          static constexpr bool is_field = false;
01138          static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01139
01143          template<typename coeffN, typename... coeffs>
01144          struct val {
01146              using enclosing_type = polynomial<Ring>;
01148              static constexpr size_t degree = sizeof...(coeffs);
01150              using aN = coeffN;
01152              using strip = val<coeffs...>;
01154              using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01156              static constexpr bool is_zero_v = is_zero_t::value;
01157
01158           private:
01159              template<size_t index, typename E = void>
01160              struct coeff_at {};
01161
01162              template<size_t index>
01163              struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))» {
01164                  using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01165              };
01166
01167              template<size_t index>
01168              struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))» {
01169                  using type = typename Ring::zero;
01170              };
01171
01172           public:
01175              template<size_t index>
01176              using coeff_at_t = typename coeff_at<index>::type;
01177
01180              static std::string to_string() {
01181                  return string_helper<coeffN, coeffs...>::func();
01182              }
01183
01188              template<typename valueRing>
01189              static constexpr valueRing eval(const valueRing& x) {
01190                  return horner_evaluation<valueRing, val>
01191                              ::template inner<0, degree + 1>
01192                              ::func(static_cast<valueRing>(0), x);
01193              }
01194          };
01195
01198          template<typename coeffN>
01199          struct val<coeffN> {
01201              using enclosing_type = polynomial<Ring>;
01203              static constexpr size_t degree = 0;
01204              using aN = coeffN;
01205              using strip = val<coeffN>;
01206              using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01207
01208              static constexpr bool is_zero_v = is_zero_t::value;
01209
01210              template<size_t index, typename E = void>
01211              struct coeff_at {};
01212
01213              template<size_t index>
01214              struct coeff_at<index, std::enable_if_t<(index == 0)» {
01215                  using type = aN;
01216              };
01217
01218              template<size_t index>
01219              struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01220                  using type = typename Ring::zero;
01221              };
01222
01223              template<size_t index>
01224              using coeff_at_t = typename coeff_at<index>::type;
01225
01226              static std::string to_string() {
01227                  return string_helper<coeffN>::func();
```

```
01228                    }
01229
01230                    template<typename valueRing>
01231                    static constexpr valueRing eval(const valueRing& x) {
01232                        return static_cast<valueRing>(aN::template get<valueRing>());
01233                    }
01234                };
01235
01237            using zero = val<typename Ring::zero>;
01239            using one = val<typename Ring::one>;
01241            using X = val<typename Ring::one, typename Ring::zero>;
01242
01243        private:
01244            template<typename P, typename E = void>
01245            struct simplify;
01246
01247            template <typename P1, typename P2, typename I>
01248            struct add_low;
01249
01250            template<typename P1, typename P2>
01251            struct add {
01252                using type = typename simplify<typename add_low<
01253                P1,
01254                P2,
01255                internal::make_index_sequence_reverse<
01256                std::max(P1::degree, P2::degree) + 1
01257                »::type>::type;
01258            };
01259
01260            template <typename P1, typename P2, typename I>
01261            struct sub_low;
01262
01263            template <typename P1, typename P2, typename I>
01264            struct mul_low;
01265
01266            template<typename v1, typename v2>
01267            struct mul {
01268                    using type = typename mul_low<
01269                        v1,
01270                        v2,
01271                        internal::make_index_sequence_reverse<
01272                        v1::degree + v2::degree + 1
01273                        »::type;
01274            };
01275
01276            template<typename coeff, size_t deg>
01277            struct monomial;
01278
01279            template<typename v, typename E = void>
01280            struct derive_helper {};
01281
01282            template<typename v>
01283            struct derive_helper<v, std::enable_if_t<v::degree == 0» {
01284                using type = zero;
01285            };
01286
01287            template<typename v>
01288            struct derive_helper<v, std::enable_if_t<v::degree != 0» {
01289                using type = typename add<
01290                    typename derive_helper<typename simplify<typename v::strip>::type>::type,
01291                    typename monomial<
01292                        typename Ring::template mul_t<
01293                            typename v::aN,
01294                            typename Ring::template inject_constant_t<(v::degree)>
01295                        >,
01296                        v::degree - 1
01297                    >::type
01298                >::type;
01299            };
01300
01301            template<typename v1, typename v2, typename E = void>
01302            struct eq_helper {};
01303
01304            template<typename v1, typename v2>
01305            struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
01306                using type = std::false_type;
01307            };
01308
01309
01310            template<typename v1, typename v2>
01311            struct eq_helper<v1, v2, std::enable_if_t<
01312                v1::degree == v2::degree &&
01313                (v1::degree != 0 || v2::degree != 0) &&
01314                std::is_same<
01315                typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01316                std::false_type
01317                >::value
```

```
01318                >
01319                > {
01320                    using type = std::false_type;
01321                };
01322
01323            template<typename v1, typename v2>
01324            struct eq_helper<v1, v2, std::enable_if_t<
01325                v1::degree == v2::degree &&
01326                (v1::degree != 0 || v2::degree != 0) &&
01327                std::is_same<
01328                typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
01329                std::true_type
01330                >::value
01331            > {
01332                    using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
01333            };
01334
01335            template<typename v1, typename v2>
01336            struct eq_helper<v1, v2, std::enable_if_t<
01337                v1::degree == v2::degree &&
01338                (v1::degree == 0)
01339            > {
01340                    using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
01341            };
01342
01343            template<typename v1, typename v2, typename E = void>
01344            struct lt_helper {};
01345
01346            template<typename v1, typename v2>
01347            struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)> {
01348                    using type = std::true_type;
01349            };
01350
01351            template<typename v1, typename v2>
01352            struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)> {
01353                    using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
01354            };
01355
01356            template<typename v1, typename v2>
01357            struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)> {
01358                    using type = std::false_type;
01359            };
01360
01361            template<typename v1, typename v2, typename E = void>
01362            struct gt_helper {};
01363
01364            template<typename v1, typename v2>
01365            struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)> {
01366                    using type = std::true_type;
01367            };
01368
01369            template<typename v1, typename v2>
01370            struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)> {
01371                    using type = std::false_type;
01372            };
01373
01374            template<typename v1, typename v2>
01375            struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)> {
01376                    using type = std::false_type;
01377            };
01378
01379            // when high power is zero : strip
01380            template<typename P>
01381            struct simplify<P, std::enable_if_t<
01382                std::is_same<
01383                typename Ring::zero,
01384                typename P::aN
01385                >::value && (P::degree > 0)
01386            > {
01387                    using type = typename simplify<typename P::strip>::type;
01388            };
01389
01390            // otherwise : do nothing
01391            template<typename P>
01392            struct simplify<P, std::enable_if_t<
01393                !std::is_same<
01394                typename Ring::zero,
01395                typename P::aN
01396                >::value && (P::degree > 0)
01397            > {
01398                    using type = P;
01399            };
01400
01401            // do not simplify constants
01402            template<typename P>
01403            struct simplify<P, std::enable_if_t<P::degree == 0> {
01404                    using type = P;
```

```
01405            };
01406
01407            // addition at
01408            template<typename P1, typename P2, size_t index>
01409            struct add_at {
01410                using type =
01411                    typename Ring::template add_t<
01412                        typename P1::template coeff_at_t<index>,
01413                        typename P2::template coeff_at_t<index>>;
01414            };
01415
01416            template<typename P1, typename P2, size_t index>
01417            using add_at_t = typename add_at<P1, P2, index>::type;
01418
01419            template<typename P1, typename P2, std::size_t... I>
01420            struct add_low<P1, P2, std::index_sequence<I...>> {
01421                using type = val<add_at_t<P1, P2, I>...>;
01422            };
01423
01424            // substraction at
01425            template<typename P1, typename P2, size_t index>
01426            struct sub_at {
01427                using type =
01428                    typename Ring::template sub_t<
01429                        typename P1::template coeff_at_t<index>,
01430                        typename P2::template coeff_at_t<index>>;
01431            };
01432
01433            template<typename P1, typename P2, size_t index>
01434            using sub_at_t = typename sub_at<P1, P2, index>::type;
01435
01436            template<typename P1, typename P2, std::size_t... I>
01437            struct sub_low<P1, P2, std::index_sequence<I...>> {
01438                using type = val<sub_at_t<P1, P2, I>...>;
01439            };
01440
01441            template<typename P1, typename P2>
01442            struct sub {
01443                using type = typename simplify<typename sub_low<
01444                P1,
01445                P2,
01446                internal::make_index_sequence_reverse<
01447                std::max(P1::degree, P2::degree) + 1
01448                >>::type>::type;
01449            };
01450
01451            // multiplication at
01452            template<typename v1, typename v2, size_t k, size_t index, size_t stop>
01453            struct mul_at_loop_helper {
01454                using type = typename Ring::template add_t<
01455                    typename Ring::template mul_t<
01456                    typename v1::template coeff_at_t<index>,
01457                    typename v2::template coeff_at_t<k - index>
01458                    >,
01459                    typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
01460                >;
01461            };
01462
01463            template<typename v1, typename v2, size_t k, size_t stop>
01464            struct mul_at_loop_helper<v1, v2, k, stop, stop> {
01465                using type = typename Ring::template mul_t<
01466                    typename v1::template coeff_at_t<stop>,
01467                    typename v2::template coeff_at_t<0>>;
01468            };
01469
01470            template <typename v1, typename v2, size_t k, typename E = void>
01471            struct mul_at {};
01472
01473            template<typename v1, typename v2, size_t k>
01474            struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)>> {
01475                using type = typename Ring::zero;
01476            };
01477
01478            template<typename v1, typename v2, size_t k>
01479            struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)>> {
01480                using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
01481            };
01482
01483            template<typename P1, typename P2, size_t index>
01484            using mul_at_t = typename mul_at<P1, P2, index>::type;
01485
01486            template<typename P1, typename P2, std::size_t... I>
01487            struct mul_low<P1, P2, std::index_sequence<I...>> {
01488                using type = val<mul_at_t<P1, P2, I>...>;
01489            };
01490
01491            // division helper
```

```
01492          template< typename A, typename B, typename Q, typename R, typename E = void>
01493          struct div_helper {};
01494
01495          template<typename A, typename B, typename Q, typename R>
01496          struct div_helper<A, B, Q, R, std::enable_if_t<
01497              (R::degree < B::degree) ||
01498              (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
01499              using q_type = Q;
01500              using mod_type = R;
01501              using gcd_type = B;
01502          };
01503
01504          template<typename A, typename B, typename Q, typename R>
01505          struct div_helper<A, B, Q, R, std::enable_if_t<
01506              (R::degree >= B::degree) &&
01507              !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
01508           private: // NOLINT
01509              using rN = typename R::aN;
01510              using bN = typename B::aN;
01511              using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
    B::degree>::type;
01512              using rr = typename sub<R, typename mul<pT, B>::type>::type;
01513              using qq = typename add<Q, pT>::type;
01514
01515           public:
01516              using q_type = typename div_helper<A, B, qq, rr>::q_type;
01517              using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
01518              using gcd_type = rr;
01519          };
01520
01521          template<typename A, typename B>
01522          struct div {
01523              static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
01524              using q_type = typename div_helper<A, B, zero, A>::q_type;
01525              using m_type = typename div_helper<A, B, zero, A>::mod_type;
01526          };
01527
01528          template<typename P>
01529          struct make_unit {
01530              using type = typename div<P, val<typename P::aN>>::q_type;
01531          };
01532
01533          template<typename coeff, size_t deg>
01534          struct monomial {
01535              using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
01536          };
01537
01538          template<typename coeff>
01539          struct monomial<coeff, 0> {
01540              using type = val<coeff>;
01541          };
01542
01543          template<typename valueRing, typename P>
01544          struct horner_evaluation {
01545              template<size_t index, size_t stop>
01546              struct inner {
01547                  static constexpr valueRing func(const valueRing& accum, const valueRing& x) {
01548                      constexpr valueRing coeff =
01549                          static_cast<valueRing>(P::template coeff_at_t<P::degree - index>::template
    get<valueRing>());
01550                      return horner_evaluation<valueRing, P>::template inner<index + 1, stop>::func(x *
    accum + coeff, x);
01551                  }
01552              };
01553
01554              template<size_t stop>
01555              struct inner<stop, stop> {
01556                  static constexpr valueRing func(const valueRing& accum, const valueRing& x) {
01557                      return accum;
01558                  }
01559              };
01560          };
01561
01562          template<typename coeff, typename... coeffs>
01563          struct string_helper {
01564              static std::string func() {
01565                  std::string tail = string_helper<coeffs...>::func();
01566                  std::string result = "";
01567                  if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
01568                      return tail;
01569                  } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
01570                      if (sizeof...(coeffs) == 1) {
01571                          result += "x";
01572                      } else {
01573                          result += "x^" + std::to_string(sizeof...(coeffs));
01574                      }
01575                  } else {
```

```
01576                        if (sizeof...(coeffs) == 1) {
01577                            result += coeff::to_string() + " x";
01578                        } else {
01579                            result += coeff::to_string()
01580                                    + " x^" + std::to_string(sizeof...(coeffs));
01581                        }
01582                    }
01583
01584                    if (!tail.empty()) {
01585                        result += " + " + tail;
01586                    }
01587
01588                    return result;
01589                }
01590            };
01591
01592            template<typename coeff>
01593            struct string_helper<coeff> {
01594                static std::string func() {
01595                    if (!std::is_same<coeff, typename Ring::zero>::value) {
01596                        return coeff::to_string();
01597                    } else {
01598                        return "";
01599                    }
01600                }
01601            };
01602
01603        public:
01606            template<typename P>
01607            using simplify_t = typename simplify<P>::type;
01608
01612            template<typename v1, typename v2>
01613            using add_t = typename add<v1, v2>::type;
01614
01618            template<typename v1, typename v2>
01619            using sub_t = typename sub<v1, v2>::type;
01620
01624            template<typename v1, typename v2>
01625            using mul_t = typename mul<v1, v2>::type;
01626
01630            template<typename v1, typename v2>
01631            using eq_t = typename eq_helper<v1, v2>::type;
01632
01636            template<typename v1, typename v2>
01637            using lt_t = typename lt_helper<v1, v2>::type;
01638
01642            template<typename v1, typename v2>
01643            using gt_t = typename gt_helper<v1, v2>::type;
01644
01648            template<typename v1, typename v2>
01649            using div_t = typename div<v1, v2>::q_type;
01650
01654            template<typename v1, typename v2>
01655            using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
01656
01660            template<typename coeff, size_t deg>
01661            using monomial_t = typename monomial<coeff, deg>::type;
01662
01665            template<typename v>
01666            using derive_t = typename derive_helper<v>::type;
01667
01670            template<typename v>
01671            using pos_t = typename Ring::template pos_t<typename v::aN>;
01672
01675            template<typename v>
01676            static constexpr bool pos_v = pos_t<v>::value;
01677
01681            template<typename v1, typename v2>
01682            using gcd_t = std::conditional_t<
01683                Ring::is_euclidean_domain,
01684                typename make_unit<gcd_t<polynomial<Ring>, v1, v2>>::type,
01685                void>;
01686
01690            template<auto x>
01691            using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
01692
01696            template<typename v>
01697            using inject_ring_t = val<v>;
01698        };
01699 }  // namespace aerobus
01700
01701 // fraction field
01702 namespace aerobus {
01703     namespace internal {
01704         template<typename Ring, typename E = void>
01705         requires IsEuclideanDomain<Ring>
01706         struct _FractionField {};
```

```
01707
01708            template<typename Ring>
01709            requires IsEuclideanDomain<Ring>
01710            struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain» {
01712                static constexpr bool is_field = true;
01713                static constexpr bool is_euclidean_domain = true;
01714
01715             private:
01716                template<typename val1, typename val2, typename E = void>
01717                struct to_string_helper {};
01718
01719                template<typename val1, typename val2>
01720                struct to_string_helper <val1, val2,
01721                    std::enable_if_t<
01722                    Ring::template eq_t<
01723                    val2, typename Ring::one
01724                    >::value
01725                    >
01726                > {
01727                    static std::string func() {
01728                        return val1::to_string();
01729                    }
01730                };
01731
01732                template<typename val1, typename val2>
01733                struct to_string_helper<val1, val2,
01734                    std::enable_if_t<
01735                    !Ring::template eq_t<
01736                    val2,
01737                    typename Ring::one
01738                    >::value
01739                    >
01740                > {
01741                    static std::string func() {
01742                        return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
01743                    }
01744                };
01745
01746             public:
01747                template<typename val1, typename val2>
01750                struct val {
01751
01753                    using x = val1;
01755                    using y = val2;
01757                    using is_zero_t = typename val1::is_zero_t;
01759                    static constexpr bool is_zero_v = val1::is_zero_t::value;
01760
01762                    using ring_type = Ring;
01763                    using enclosing_type = _FractionField<Ring>;
01764
01767                     static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
01768
01772                    template<typename valueType>
01773                    static constexpr valueType get() { return static_cast<valueType>(x::v) /
    static_cast<valueType>(y::v); }
01774
01777                    static std::string to_string() {
01778                        return to_string_helper<val1, val2>::func();
01779                    }
01780
01785                    template<typename valueRing>
01786                    static constexpr valueRing eval(const valueRing& v) {
01787                        return x::eval(v) / y::eval(v);
01788                    }
01789                };
01790
01792                using zero = val<typename Ring::zero, typename Ring::one>;
01794                using one = val<typename Ring::one, typename Ring::one>;
01795
01798                template<typename v>
01799                using inject_t = val<v, typename Ring::one>;
01800
01803                template<auto x>
01804                using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
    Ring::one>;
01805
01808                template<typename v>
01809                using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
01810
01812                using ring_type = Ring;
01813
01814             private:
01815                template<typename v, typename E = void>
01816                struct simplify {};
01817
01818                // x = 0
01819                template<typename v>
01820                struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {
```

```
01821                  using type = typename _FractionField<Ring>::zero;
01822               };
01823
01824              // x != 0
01825              template<typename v>
01826              struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value> {
01827               private:
01828                  using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
01829                  using newx = typename Ring::template div_t<typename v::x, _gcd>;
01830                  using newy = typename Ring::template div_t<typename v::y, _gcd>;
01831
01832                  using posx = std::conditional_t<
01833                                  !Ring::template pos_v<newy>,
01834                                  typename Ring::template sub_t<typename Ring::zero, newx>,
01835                                  newx>;
01836                  using posy = std::conditional_t<
01837                                  !Ring::template pos_v<newy>,
01838                                  typename Ring::template sub_t<typename Ring::zero, newy>,
01839                                  newy>;
01840               public:
01841                  using type = typename _FractionField<Ring>::template val<posx, posy>;
01842               };
01843
01844          public:
01847              template<typename v>
01848              using simplify_t = typename simplify<v>::type;
01849
01850          private:
01851              template<typename v1, typename v2>
01852              struct add {
01853               private:
01854                  using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01855                  using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01856                  using dividend = typename Ring::template add_t<a, b>;
01857                  using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01858                  using g = typename Ring::template gcd_t<dividend, diviser>;
01859
01860               public:
01861                  using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
      diviser»;
01862               };
01863
01864              template<typename v>
01865              struct pos {
01866                  using type = std::conditional_t<
01867                      (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
01868                      (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
01869                      std::true_type,
01870                      std::false_type>;
01871               };
01872
01873              template<typename v1, typename v2>
01874              struct sub {
01875               private:
01876                  using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01877                  using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01878                  using dividend = typename Ring::template sub_t<a, b>;
01879                  using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01880                  using g = typename Ring::template gcd_t<dividend, diviser>;
01881
01882               public:
01883                  using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
      diviser»;
01884               };
01885
01886              template<typename v1, typename v2>
01887              struct mul {
01888               private:
01889                  using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
01890                  using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01891
01892               public:
01893                  using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
01894               };
01895
01896              template<typename v1, typename v2, typename E = void>
01897              struct div {};
01898
01899              template<typename v1, typename v2>
01900              struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
      _FractionField<Ring>::zero>::value»  {
01901               private:
01902                  using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01903                  using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01904
01905               public:
01906                  using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
```

```
01907                };
01908
01909            template<typename v1, typename v2>
01910            struct div<v1, v2, std::enable_if_t<
01911                std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
01912                using type = one;
01913            };
01914
01915            template<typename v1, typename v2>
01916            struct eq {
01917                using type = std::conditional_t<
01918                        std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
01919                        std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
01920                    std::true_type,
01921                    std::false_type>;
01922            };
01923
01924            template<typename v1, typename v2, typename E = void>
01925            struct gt;
01926
01927            template<typename v1, typename v2>
01928            struct gt<v1, v2, std::enable_if_t<
01929                (eq<v1, v2>::type::value)
01930                » {
01931                using type = std::false_type;
01932            };
01933
01934            template<typename v1, typename v2>
01935            struct gt<v1, v2, std::enable_if_t<
01936                (!eq<v1, v2>::type::value) &&
01937                (!pos<v1>::type::value) && (!pos<v2>::type::value)
01938                » {
01939                using type = typename gt<
01940                    typename sub<zero, v1>::type, typename sub<zero, v2>::type
01941                >::type;
01942            };
01943
01944            template<typename v1, typename v2>
01945            struct gt<v1, v2, std::enable_if_t<
01946                (!eq<v1, v2>::type::value) &&
01947                (pos<v1>::type::value) && (!pos<v2>::type::value)
01948                » {
01949                using type = std::true_type;
01950            };
01951
01952            template<typename v1, typename v2>
01953            struct gt<v1, v2, std::enable_if_t<
01954                (!eq<v1, v2>::type::value) &&
01955                (!pos<v1>::type::value) && (pos<v2>::type::value)
01956                » {
01957                using type = std::false_type;
01958            };
01959
01960            template<typename v1, typename v2>
01961            struct gt<v1, v2, std::enable_if_t<
01962                (!eq<v1, v2>::type::value) &&
01963                (pos<v1>::type::value) && (pos<v2>::type::value)
01964                » {
01965                using type = typename Ring::template gt_t<
01966                    typename Ring::template mul_t<v1::x, v2::y>,
01967                    typename Ring::template mul_t<v2::y, v2::x>
01968                >;
01969            };
01970
01971        public:
01976            template<typename v1, typename v2>
01977            using add_t = typename add<v1, v2>::type;
01978
01983            template<typename v1, typename v2>
01984            using mod_t = zero;
01985
01990            template<typename v1, typename v2>
01991            using gcd_t = v1;
01992
01996            template<typename v1, typename v2>
01997            using sub_t = typename sub<v1, v2>::type;
01998
02002            template<typename v1, typename v2>
02003            using mul_t = typename mul<v1, v2>::type;
02004
02008            template<typename v1, typename v2>
02009            using div_t = typename div<v1, v2>::type;
02010
02014            template<typename v1, typename v2>
02015            using eq_t = typename eq<v1, v2>::type;
02016
02020            template<typename v1, typename v2>
```

```
02021            static constexpr bool eq_v = eq<v1, v2>::type::value;
02022
02026            template<typename v1, typename v2>
02027            using gt_t = typename gt<v1, v2>::type;
02028
02032            template<typename v1, typename v2>
02033            static constexpr bool gt_v = gt<v1, v2>::type::value;
02034
02037            template<typename v1>
02038            using pos_t = typename pos<v1>::type;
02039
02042            template<typename v>
02043            static constexpr bool pos_v = pos_t<v>::value;
02044        };
02045
02046        template<typename Ring, typename E = void>
02047        requires IsEuclideanDomain<Ring>
02048        struct FractionFieldImpl {};
02049
02050        // fraction field of a field is the field itself
02051        template<typename Field>
02052        requires IsEuclideanDomain<Field>
02053        struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {
02054            using type = Field;
02055            template<typename v>
02056            using inject_t = v;
02057        };
02058
02059        // fraction field of a ring is the actual fraction field
02060        template<typename Ring>
02061        requires IsEuclideanDomain<Ring>
02062        struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {
02063            using type = _FractionField<Ring>;
02064        };
02065    }  // namespace internal
02066
02070    template<typename Ring>
02071    requires IsEuclideanDomain<Ring>
02072    using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02073 }  // namespace aerobus
02074
02075 // short names for common types
02076 namespace aerobus {
02079    using q32 = FractionField<i32>;
02082    using fpq32 = FractionField<polynomial<q32>>;
02085    using q64 = FractionField<i64>;
02087    using pi64 = polynomial<i64>;
02089    using pq64 = polynomial<q64>;
02091    using fpq64 = FractionField<polynomial<q64>>;
02096    template<typename Ring, typename v1, typename v2>
02097    using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02098
02102    template<int64_t p, int64_t q>
02103    using make_q64_t = typename q64::template simplify_t<
02104            typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02105
02109    template<int32_t p, int32_t q>
02110    using make_q32_t = typename q32::template simplify_t<
02111            typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>»;
02112
02117    template<typename Ring, typename v1, typename v2>
02118    using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02123    template<typename Ring, typename v1, typename v2>
02124    using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02125 }  // namespace aerobus
02126
02127 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02128 namespace aerobus {
02129    namespace internal {
02130        template<typename T, size_t x, typename E = void>
02131        struct factorial {};
02132
02133        template<typename T, size_t x>
02134        struct factorial<T, x, std::enable_if_t<(x > 0)» {
02135        private:
02136            template<typename, size_t, typename>
02137            friend struct factorial;
02138        public:
02139            using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
    x - 1>::type>;
02140            static constexpr typename T::inner_type value = type::template get<typename
    T::inner_type>();
02141        };
02142
02143        template<typename T>
02144        struct factorial<T, 0> {
02145         public:
```

```
02146                 using type = typename T::one;
02147                 static constexpr typename T::inner_type value = type::template get<typename
        T::inner_type>();
02148             };
02149         }  // namespace internal
02150
02154         template<typename T, size_t i>
02155         using factorial_t = typename internal::factorial<T, i>::type;
02156
02160         template<typename T, size_t i>
02161         inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02162
02163         namespace internal {
02164             template<typename T, size_t k, size_t n, typename E = void>
02165             struct combination_helper {};
02166
02167             template<typename T, size_t k, size_t n>
02168             struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)>> {
02169                 using type = typename FractionField<T>::template mul_t<
02170                     typename combination_helper<T, k - 1, n - 1>::type,
02171                     makefraction_t<T, typename T::template val<n>, typename T::template val<k>>>;
02172             };
02173
02174             template<typename T, size_t k, size_t n>
02175             struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)>> {
02176                 using type = typename combination_helper<T, n - k, n>::type;
02177             };
02178
02179             template<typename T, size_t n>
02180             struct combination_helper<T, 0, n> {
02181                 using type = typename FractionField<T>::one;
02182             };
02183
02184             template<typename T, size_t k, size_t n>
02185             struct combination {
02186                 using type = typename internal::combination_helper<T, k, n>::type::x;
02187                 static constexpr typename T::inner_type value =
02188                         internal::combination_helper<T, k, n>::type::template get<typename
        T::inner_type>();
02189             };
02190         }  // namespace internal
02191
02194         template<typename T, size_t k, size_t n>
02195         using combination_t = typename internal::combination<T, k, n>::type;
02196
02201         template<typename T, size_t k, size_t n>
02202         inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02203
02204         namespace internal {
02205             template<typename T, size_t m>
02206             struct bernoulli;
02207
02208             template<typename T, typename accum, size_t k, size_t m>
02209             struct bernoulli_helper {
02210                 using type = typename bernoulli_helper<
02211                     T,
02212                     addfractions_t<T,
02213                         accum,
02214                         mulfractions_t<T,
02215                             makefraction_t<T,
02216                                 combination_t<T, k, m + 1>,
02217                                 typename T::one>,
02218                             typename bernoulli<T, k>::type
02219                         >
02220                     >,
02221                     k + 1,
02222                     m>::type;
02223             };
02224
02225             template<typename T, typename accum, size_t m>
02226             struct bernoulli_helper<T, accum, m, m> {
02227                 using type = accum;
02228             };
02229
02230
02231
02232             template<typename T, size_t m>
02233             struct bernoulli {
02234                 using type = typename FractionField<T>::template mul_t<
02235                     typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02236                     makefraction_t<T,
02237                     typename T::template val<static_cast<typename T::inner_type>(-1)>,
02238                     typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02239                     >
02240                 >;
02241
02242             template<typename floatType>
```

```
02243                   static constexpr floatType value = type::template get<floatType>();
02244              };
02245
02246          template<typename T>
02247          struct bernoulli<T, 0> {
02248              using type = typename FractionField<T>::one;
02249
02250              template<typename floatType>
02251              static constexpr floatType value = type::template get<floatType>();
02252          };
02253      }  // namespace internal
02254
02258      template<typename T, size_t n>
02259      using bernoulli_t = typename internal::bernoulli<T, n>::type;
02260
02265      template<typename FloatType, typename T, size_t n >
02266      inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02267
02268      namespace internal {
02269          template<typename T, int k, typename E = void>
02270          struct alternate {};
02271
02272          template<typename T, int k>
02273          struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
02274              using type = typename T::one;
02275              static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
02276          };
02277
02278          template<typename T, int k>
02279          struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
02280              using type = typename T::template sub_t<typename T::zero, typename T::one>;
02281              static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
02282          };
02283      }  // namespace internal
02284
02287      template<typename T, int k>
02288      using alternate_t = typename internal::alternate<T, k>::type;
02289
02290      namespace internal {
02291          template<typename T, int n, int k, typename E = void>
02292          struct stirling_helper {};
02293
02294          template<typename T>
02295          struct stirling_helper<T, 0, 0> {
02296              using type = typename T::one;
02297          };
02298
02299          template<typename T, int n>
02300          struct stirling_helper<T, n, 0, std::enable_if_t<(n > 0)» {
02301              using type = typename T::zero;
02302          };
02303
02304          template<typename T, int n>
02305          struct stirling_helper<T, 0, n, std::enable_if_t<(n > 0)» {
02306              using type = typename T::zero;
02307          };
02308
02309          template<typename T, int n, int k>
02310          struct stirling_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)» {
02311              using type = typename T::template sub_t<
02312                               typename stirling_helper<T, n-1, k-1>::type,
02313                               typename T::template mul_t<
02314                                   typename T::template inject_constant_t<n-1>,
02315                                   typename stirling_helper<T, n-1, k>::type
02316                               »;
02317          };
02318      }  // namespace internal
02319
02324      template<typename T, int n, int k>
02325      using stirling_signed_t = typename internal::stirling_helper<T, n, k>::type;
02326
02331      template<typename T, int n, int k>
02332      using stirling_unsigned_t = abs_t<typename internal::stirling_helper<T, n, k>::type>;
02333
02338      template<typename T, int n, int k>
02339      static constexpr typename T::inner_type stirling_signed_v = stirling_signed_t<T, n, k>::v;
02340
02341
02346      template<typename T, int n, int k>
02347      static constexpr typename T::inner_type stirling_unsigned_v = stirling_unsigned_t<T, n, k>::v;
02348
02351      template<typename T, size_t k>
02352      inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
02353
02354      namespace internal {
```

```
02355          template<typename T, auto p, auto n, typename E = void>
02356          struct pow {};
02357
02358          template<typename T, auto p, auto n>
02359          struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)» {
02360              using type = typename T::template mul_t<
02361                  typename pow<T, p, n/2>::type,
02362                  typename pow<T, p, n/2>::type
02363              >;
02364          };
02365
02366          template<typename T, auto p, auto n>
02367          struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)» {
02368              using type = typename T::template mul_t<
02369                  typename T::template inject_constant_t<p>,
02370                  typename T::template mul_t<
02371                      typename pow<T, p, n/2>::type,
02372                      typename pow<T, p, n/2>::type
02373                  >
02374              >;
02375          };
02376
02377          template<typename T, auto n, auto p>
02378          struct pow<T, n, p, std::enable_if_t<p == 0» { using type = typename T::one; };
02379      }  // namespace internal
02380
02385      template<typename T, auto p, auto n>
02386      using pow_t = typename internal::pow<T, p, n>::type;
02387
02392      template<typename T, auto p, auto n>
02393      static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
02394
02395      namespace internal {
02396          template<typename, template<typename, size_t> typename, class>
02397          struct make_taylor_impl;
02398
02399          template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
02400          struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...» {
02401              using type = typename polynomial<FractionField<T>>::template val<typename coeff_at<T,
       Is>::type...>;
02402          };
02403      }
02404
02409      template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
02410      using taylor = typename internal::make_taylor_impl<
02411          T,
02412          coeff_at,
02413          internal::make_index_sequence_reverse<deg + 1>>::type;
02414
02415      namespace internal {
02416          template<typename T, size_t i>
02417          struct exp_coeff {
02418              using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02419          };
02420
02421          template<typename T, size_t i, typename E = void>
02422          struct sin_coeff_helper {};
02423
02424          template<typename T, size_t i>
02425          struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02426              using type = typename FractionField<T>::zero;
02427          };
02428
02429          template<typename T, size_t i>
02430          struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02431              using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02432          };
02433
02434          template<typename T, size_t i>
02435          struct sin_coeff {
02436              using type = typename sin_coeff_helper<T, i>::type;
02437          };
02438
02439          template<typename T, size_t i, typename E = void>
02440          struct sh_coeff_helper {};
02441
02442          template<typename T, size_t i>
02443          struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02444              using type = typename FractionField<T>::zero;
02445          };
02446
02447          template<typename T, size_t i>
02448          struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02449              using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02450          };
02451
02452          template<typename T, size_t i>
```

```
02453            struct sh_coeff {
02454                using type = typename sh_coeff_helper<T, i>::type;
02455            };
02456
02457            template<typename T, size_t i, typename E = void>
02458            struct cos_coeff_helper {};
02459
02460            template<typename T, size_t i>
02461            struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02462                using type = typename FractionField<T>::zero;
02463            };
02464
02465            template<typename T, size_t i>
02466            struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02467                using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02468            };
02469
02470            template<typename T, size_t i>
02471            struct cos_coeff {
02472                using type = typename cos_coeff_helper<T, i>::type;
02473            };
02474
02475            template<typename T, size_t i, typename E = void>
02476            struct cosh_coeff_helper {};
02477
02478            template<typename T, size_t i>
02479            struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02480                using type = typename FractionField<T>::zero;
02481            };
02482
02483            template<typename T, size_t i>
02484            struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02485                using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02486            };
02487
02488            template<typename T, size_t i>
02489            struct cosh_coeff {
02490                using type = typename cosh_coeff_helper<T, i>::type;
02491            };
02492
02493            template<typename T, size_t i>
02494            struct geom_coeff { using type = typename FractionField<T>::one; };
02495
02496
02497            template<typename T, size_t i, typename E = void>
02498            struct atan_coeff_helper;
02499
02500            template<typename T, size_t i>
02501            struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02502                using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i»;
02503            };
02504
02505            template<typename T, size_t i>
02506            struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02507                using type = typename FractionField<T>::zero;
02508            };
02509
02510            template<typename T, size_t i>
02511            struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
02512
02513            template<typename T, size_t i, typename E = void>
02514            struct asin_coeff_helper;
02515
02516            template<typename T, size_t i>
02517            struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02518                using type = makefraction_t<T,
02519                    factorial_t<T, i - 1>,
02520                    typename T::template mul_t<
02521                        typename T::template val<i>,
02522                        T::template mul_t<
02523                            pow_t<T, 4, i / 2>,
02524                            pow<T, factorial<T, i / 2>::value, 2
02525                        >
02526                    >
02527                »;
02528            };
02529
02530            template<typename T, size_t i>
02531            struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02532                using type = typename FractionField<T>::zero;
02533            };
02534
02535            template<typename T, size_t i>
02536            struct asin_coeff {
02537                using type = typename asin_coeff_helper<T, i>::type;
02538            };
02539
```

```
02540            template<typename T, size_t i>
02541            struct lnp1_coeff {
02542                using type = makefraction_t<T,
02543                    alternate_t<T, i + 1>,
02544                    typename T::template val<i»;
02545            };
02546
02547            template<typename T>
02548            struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
02549
02550            template<typename T, size_t i, typename E = void>
02551            struct asinh_coeff_helper;
02552
02553            template<typename T, size_t i>
02554            struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02555                using type = makefraction_t<T,
02556                    typename T::template mul_t<
02557                        alternate_t<T, i / 2>,
02558                        factorial_t<T, i - 1>
02559                    >,
02560                    typename T::template mul_t<
02561                        typename T::template mul_t<
02562                            typename T::template val<i>,
02563                            pow_t<T, (factorial<T, i / 2>::value), 2>
02564                        >,
02565                        pow_t<T, 4, i / 2>
02566                    >
02567                >;
02568            };
02569
02570            template<typename T, size_t i>
02571            struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02572                using type = typename FractionField<T>::zero;
02573            };
02574
02575            template<typename T, size_t i>
02576            struct asinh_coeff {
02577                using type = typename asinh_coeff_helper<T, i>::type;
02578            };
02579
02580            template<typename T, size_t i, typename E = void>
02581            struct atanh_coeff_helper;
02582
02583            template<typename T, size_t i>
02584            struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02585                // 1/i
02586                using type = typename FractionField<T>:: template val<
02587                    typename T::one,
02588                    typename T::template inject_constant_t<i»;
02589            };
02590
02591            template<typename T, size_t i>
02592            struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
02593                using type = typename FractionField<T>::zero;
02594            };
02595
02596            template<typename T, size_t i>
02597            struct atanh_coeff {
02598                using type = typename atanh_coeff_helper<T, i>::type;
02599            };
02600
02601            template<typename T, size_t i, typename E = void>
02602            struct tan_coeff_helper;
02603
02604            template<typename T, size_t i>
02605            struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
02606                using type = typename FractionField<T>::zero;
02607            };
02608
02609            template<typename T, size_t i>
02610            struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
02611            private:
02612                // 4^((i+1)/2)
02613                using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
02614                // 4^((i+1)/2) - 1
02615                using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
       FractionField<T>::one>;
02616                // (-1)^((i-1)/2)
02617                using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
02618                using dividend = typename FractionField<T>::template mul_t<
02619                    altp,
02620                    FractionField<T>::template mul_t<
02621                    _4p,
02622                    FractionField<T>::template mul_t<
02623                    _4pm1,
02624                    bernoulli_t<T, (i + 1)>
02625                    >
```

```
02626                        >
02627                    >;
02628            public:
02629                using type = typename FractionField<T>::template div_t<dividend,
02630                    typename FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
02631            };
02632
02633            template<typename T, size_t i>
02634            struct tan_coeff {
02635                using type = typename tan_coeff_helper<T, i>::type;
02636            };
02637
02638            template<typename T, size_t i, typename E = void>
02639            struct tanh_coeff_helper;
02640
02641            template<typename T, size_t i>
02642            struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
02643                using type = typename FractionField<T>::zero;
02644            };
02645
02646            template<typename T, size_t i>
02647            struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
02648            private:
02649                using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
02650                using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
        FractionField<T>::one>;
02651                using dividend =
02652                    typename FractionField<T>::template mul_t<
02653                        _4p,
02654                        typename FractionField<T>::template mul_t<
02655                            _4pm1,
02656                            bernoulli_t<T, (i + 1)»>::type;
02657            public:
02658                using type = typename FractionField<T>::template div_t<dividend,
02659                    FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
02660            };
02661
02662            template<typename T, size_t i>
02663            struct tanh_coeff {
02664                using type = typename tanh_coeff_helper<T, i>::type;
02665            };
02666        }  // namespace internal
02667
02671        template<typename Integers, size_t deg>
02672        using exp = taylor<Integers, internal::exp_coeff, deg>;
02673
02677        template<typename Integers, size_t deg>
02678        using expm1 = typename polynomial<FractionField<Integers>>::template sub_t<
02679            exp<Integers, deg>,
02680            typename polynomial<FractionField<Integers>>::one>;
02681
02685        template<typename Integers, size_t deg>
02686        using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
02687
02691        template<typename Integers, size_t deg>
02692        using atan = taylor<Integers, internal::atan_coeff, deg>;
02693
02697        template<typename Integers, size_t deg>
02698        using sin = taylor<Integers, internal::sin_coeff, deg>;
02699
02703        template<typename Integers, size_t deg>
02704        using sinh = taylor<Integers, internal::sh_coeff, deg>;
02705
02710        template<typename Integers, size_t deg>
02711        using cosh = taylor<Integers, internal::cosh_coeff, deg>;
02712
02717        template<typename Integers, size_t deg>
02718        using cos = taylor<Integers, internal::cos_coeff, deg>;
02719
02724        template<typename Integers, size_t deg>
02725        using geometric_sum = taylor<Integers, internal::geom_coeff, deg»;
02726
02731        template<typename Integers, size_t deg>
02732        using asin = taylor<Integers, internal::asin_coeff, deg>;
02733
02738        template<typename Integers, size_t deg>
02739        using asinh = taylor<Integers, internal::asinh_coeff, deg>;
02740
02745        template<typename Integers, size_t deg>
02746        using atanh = taylor<Integers, internal::atanh_coeff, deg>;
02747
02752        template<typename Integers, size_t deg>
02753        using tan = taylor<Integers, internal::tan_coeff, deg>;
02754
02759        template<typename Integers, size_t deg>
02760        using tanh = taylor<Integers, internal::tanh_coeff, deg>;
02761 }  // namespace aerobus
```

```
02762
02763  // continued fractions
02764  namespace aerobus {
02773      template<int64_t... values>
02774      struct ContinuedFraction {};
02775
02778      template<int64_t a0>
02779      struct ContinuedFraction<a0> {
02781          using type = typename q64::template inject_constant_t<a0>;
02783          static constexpr double val = static_cast<double>(a0);
02784      };
02785
02789      template<int64_t a0, int64_t... rest>
02790      struct ContinuedFraction<a0, rest...> {
02792          using type = q64::template add_t<
02793                  typename q64::template inject_constant_t<a0>,
02794                  typename q64::template div_t<
02795                      typename q64::one,
02796                      typename ContinuedFraction<rest...>::type
02797                  >>;
02799          static constexpr double val = type::template get<double>();
02800      };
02801
02806      using PI_fraction =
02      ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1>;
02809      using E_fraction =
02      ContinuedFraction<2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1>;
02811      using SQRT2_fraction =
02      ContinuedFraction<1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2>;
02813      using SQRT3_fraction =
02      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>;
02      // NOLINT
02814  }  // namespace aerobus
02815
02816  // known polynomials
02817  namespace aerobus {
02818      // CChebyshev
02819      namespace internal {
02820          template<int kind, size_t deg>
02821          struct chebyshev_helper {
02822              using type = typename pi64::template sub_t<
02823                  typename pi64::template mul_t<
02824                      typename pi64::template mul_t<
02825                          pi64::inject_constant_t<2>,
02826                          typename pi64::X>,
02827                      typename chebyshev_helper<kind, deg - 1>::type
02828                  >,
02829                  typename chebyshev_helper<kind, deg - 2>::type
02830              >;
02831          };
02832
02833          template<>
02834          struct chebyshev_helper<1, 0> {
02835              using type = typename pi64::one;
02836          };
02837
02838          template<>
02839          struct chebyshev_helper<1, 1> {
02840              using type = typename pi64::X;
02841          };
02842
02843          template<>
02844          struct chebyshev_helper<2, 0> {
02845              using type = typename pi64::one;
02846          };
02847
02848          template<>
02849          struct chebyshev_helper<2, 1> {
02850              using type = typename pi64::template mul_t<
02851                  typename pi64::inject_constant_t<2>,
02852                  typename pi64::X>;
02853          };
02854      }  // namespace internal
02855
02856      // Laguerre
02857      namespace internal {
02858          template<size_t deg>
02859          struct laguerre_helper {
02860           private:
02861              // Lk = (1 / k) * ((2 * k - 1 - x) * lkm1 - (k - 2)Lkm2)
02862              using lnm2 = typename laguerre_helper<deg - 2>::type;
02863              using lnm1 = typename laguerre_helper<deg - 1>::type;
02864              // -x + 2k-1
02865              using p = typename pq64::template val<
02866                  typename q64::template inject_constant_t<-1>,
02867                  typename q64::template inject_constant_t<2 * deg - 1>>;
02868              // 1/n
```

```
02869                using factor = typename pq64::template inject_ring_t<
02870                    q64::val<typename i64::one, typename i64::template inject_constant_t<deg»>;
02871
02872            public:
02873                using type = typename pq64::template mul_t <
02874                    factor,
02875                    typename pq64::template sub_t<
02876                        typename pq64::template mul_t<
02877                            p,
02878                            lnm1
02879                        >,
02880                        typename pq64::template mul_t<
02881                            typename pq64::template inject_constant_t<deg-1>,
02882                            lnm2
02883                        >
02884                    >
02885                >;
02886            };
02887
02888            template<>
02889            struct laguerre_helper<0> {
02890                using type = typename pq64::one;
02891            };
02892
02893            template<>
02894            struct laguerre_helper<1> {
02895                using type = typename pq64::template sub_t<typename pq64::one, typename pq64::X>;
02896            };
02897        }  // namespace internal
02898
02899        // Bernstein
02900        namespace internal {
02901            template<size_t i, size_t m, typename E = void>
02902            struct bernstein_helper {};
02903
02904            template<>
02905            struct bernstein_helper<0, 0> {
02906                using type = typename pi64::one;
02907            };
02908
02909            template<size_t i, size_t m>
02910            struct bernstein_helper<i, m, std::enable_if_t<
02911                    (m > 0) && (i == 0)» {
02912                using type = typename pi64::mul_t<
02913                    typename pi64::sub_t<typename pi64::one, typename pi64::X>,
02914                    typename bernstein_helper<i, m-1>::type>;
02915            };
02916
02917            template<size_t i, size_t m>
02918            struct bernstein_helper<i, m, std::enable_if_t<
02919                    (m > 0) && (i == m)» {
02920                using type = typename pi64::template mul_t<
02921                    typename pi64::X,
02922                    typename bernstein_helper<i-1, m-1>::type>;
02923            };
02924
02925            template<size_t i, size_t m>
02926            struct bernstein_helper<i, m, std::enable_if_t<
02927                    (m > 0) && (i > 0) && (i < m)» {
02928                using type = typename pi64::add_t<
02929                    typename pi64::mul_t<
02930                        typename pi64::sub_t<typename pi64::one, typename pi64::X>,
02931                        typename bernstein_helper<i, m-1>::type>,
02932                    typename pi64::mul_t<
02933                        typename pi64::X,
02934                        typename bernstein_helper<i-1, m-1>::type»;
02935            };
02936        }  // namespace internal
02937
02938        namespace known_polynomials {
02940            enum hermite_kind {
02942                probabilist,
02944                physicist
02945            };
02946        }
02947
02948        // hermite
02949        namespace internal {
02950            template<size_t deg, known_polynomials::hermite_kind kind>
02951            struct hermite_helper {};
02952
02953            template<size_t deg>
02954            struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist> {
02955            private:
02956                using hnm1 = typename hermite_helper<deg - 1,
    known_polynomials::hermite_kind::probabilist>::type;
02957                using hnm2 = typename hermite_helper<deg - 2,
```

```
              known_polynomials::hermite_kind::probabilist>::type;
02958
02959            public:
02960              using type = typename pi64::template sub_t<
02961                  typename pi64::template mul_t<typename pi64::X, hnm1>,
02962                  typename pi64::template mul_t<
02963                      typename pi64::template inject_constant_t<deg - 1>,
02964                      hnm2
02965                  >
02966              >;
02967          };
02968
02969          template<size_t deg>
02970          struct hermite_helper<deg, known_polynomials::hermite_kind::physicist> {
02971           private:
02972              using hnm1 = typename hermite_helper<deg - 1,
              known_polynomials::hermite_kind::physicist>::type;
02973              using hnm2 = typename hermite_helper<deg - 2,
              known_polynomials::hermite_kind::physicist>::type;
02974
02975           public:
02976              using type = typename pi64::template sub_t<
02977                  // 2X Hn-1
02978                  typename pi64::template mul_t<
02979                      typename pi64::val<typename i64::template inject_constant_t<2>,
02980                      typename i64::zero>, hnm1>,
02981
02982                  typename pi64::template mul_t<
02983                      typename pi64::template inject_constant_t<2*(deg - 1)>,
02984                      hnm2
02985                  >
02986              >;
02987          };
02988
02989          template<>
02990          struct hermite_helper<0, known_polynomials::hermite_kind::probabilist> {
02991              using type = typename pi64::one;
02992          };
02993
02994          template<>
02995          struct hermite_helper<1, known_polynomials::hermite_kind::probabilist> {
02996              using type = typename pi64::X;
02997          };
02998
02999          template<>
03000          struct hermite_helper<0, known_polynomials::hermite_kind::physicist> {
03001              using type = typename pi64::one;
03002          };
03003
03004          template<>
03005          struct hermite_helper<1, known_polynomials::hermite_kind::physicist> {
03006              // 2X
03007              using type = typename pi64::template val<typename i64::template inject_constant_t<2>,
              typename i64::zero>;
03008          };
03009      }  // namespace internal
03010
03011      // legendre
03012      namespace internal {
03013          template<size_t n>
03014          struct legendre_helper {
03015           private:
03016              // 1/n constant
03017              // (2n-1)/n X
03018              using fact_left = typename pq64::monomial_t<make_q64_t<2*n-1, n>, 1>;
03019              // (n-1) / n
03020              using fact_right = typename pq64::val<make_q64_t<n-1, n>>;
03021           public:
03022              using type = pq64::template sub_t<
03023                  typename pq64::template mul_t<
03024                      fact_left,
03025                      typename legendre_helper<n-1>::type
03026                  >,
03027                  typename pq64::template mul_t<
03028                      fact_right,
03029                      typename legendre_helper<n-2>::type
03030                  >
03031              >;
03032          };
03033
03034          template<>
03035          struct legendre_helper<0> {
03036              using type = typename pq64::one;
03037          };
03038
03039          template<>
03040          struct legendre_helper<1> {
```

```
03041                using type = typename pq64::X;
03042            };
03043        }  // namespace internal
03044
03045        // bernoulli polynomials
03046        namespace internal {
03047            template<size_t n>
03048            struct bernoulli_coeff {
03049                template<typename T, size_t i>
03050                struct inner {
03051                 private:
03052                    using F = FractionField<T>;
03053                 public:
03054                    using type = typename F::template mul_t<
03055                        typename F::template inject_ring_t<combination_t<T, i, n>>,
03056                        bernoulli_t<T, n-i>
03057                    >;
03058                };
03059            };
03060        }  // namespace internal
03061
03063        namespace known_polynomials {
03070            template <size_t deg>
03071            using chebyshev_T = typename internal::chebyshev_helper<1, deg>::type;
03072
03079            template <size_t deg>
03080            using chebyshev_U = typename internal::chebyshev_helper<2, deg>::type;
03081
03088            template <size_t deg>
03089            using laguerre = typename internal::laguerre_helper<deg>::type;
03090
03097            template <size_t deg>
03098            using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist>::type;
03099
03106            template <size_t deg>
03107            using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist>::type;
03108
03116            template<size_t i, size_t m>
03117            using bernstein = typename internal::bernstein_helper<i, m>::type;
03118
03125            template<size_t deg>
03126            using legendre = typename internal::legendre_helper<deg>::type;
03127
03134            template<size_t deg>
03135            using bernoulli = taylor<i64, internal::bernoulli_coeff<deg>::template inner, deg>;
03136        }  // namespace known_polynomials
03137 }  // namespace aerobus
03138
03139
03140 #ifdef AEROBUS_CONWAY_IMPORTS
03141
03142 // conway polynomials
03143 namespace aerobus {
03147        template<int p, int n>
03148        struct ConwayPolynomial {};
03149
03150 #ifndef DO_NOT_DOCUMENT
03151        #define ZPZV ZPZ::template val
03152        #define POLYV aerobus::polynomial<ZPZ>::template val
03153        template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
    POLYV<ZPZV<1>, ZPZV<1>; };  // NOLINT
03154        template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
    POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>; }; // NOLINT
03155        template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>; };  // NOLINT
03156        template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; };  // NOLINT
03157        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
03158        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
03159        template<> struct ConwayPolynomial<2, 7> { using ZPZ = aerobus::zpz<2>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; };  // NOLINT
03160        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
03161        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
03162        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
03163        template<> struct ConwayPolynomial<2, 11> { using ZPZ = aerobus::zpz<2>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
    ZPZV<0>, ZPZV<1>; };  // NOLINT
03164        template<> struct ConwayPolynomial<2, 12> { 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>,
    ZPZV<0>, ZPZV<1>, ZPZV<1>; };  // NOLINT
```

```
03165    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<0>, ZPZV<1>,
     ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>; };  // NOLINT
03166    template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>,
     ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>; };  // NOLINT
03167    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<0>,
     ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>; };  // NOLINT
03168    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<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>; };  // NOLINT
03169    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<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>; };  // NOLINT
03170    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
03171    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
03172    template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
     ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; };
     // NOLINT
03173    template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
     POLYV<ZPZV<1>, ZPZV<1>; };  // NOLINT
03174    template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
     POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>; };  // NOLINT
03175    template<> struct ConwayPolynomial<3, 3> { using ZPZ = aerobus::zpz<3>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<1>; };  // NOLINT
03176    template<> struct ConwayPolynomial<3, 4> { using ZPZ = aerobus::zpz<3>; using type =
     POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2>; };  // NOLINT
03177    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
03178    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
03179    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
03180    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
03181    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
03182    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
03183    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
03184    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<2>; };  // NOLINT
03185    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
03186    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
03187    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
03188    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
03189    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
03190    template<> struct ConwayPolynomial<3, 18> { using ZPZ = aerobus::zpz<3>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>,
     ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>; };  // NOLINT
03191    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
03192    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
03193    template<> struct ConwayPolynomial<5, 1> { using ZPZ = aerobus::zpz<5>; using type =
     POLYV<ZPZV<1>, ZPZV<3>; };  // NOLINT
03194    template<> struct ConwayPolynomial<5, 2> { using ZPZ = aerobus::zpz<5>; using type =
     POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2>; };  // NOLINT
03195    template<> struct ConwayPolynomial<5, 3> { using ZPZ = aerobus::zpz<5>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3>; };  // NOLINT
03196    template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
```

```
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; };  // NOLINT
03197     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
03198     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
03199     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
03200     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
03201     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
03202     template<> struct ConwayPolynomial<5, 10> { using ZPZ = aerobus::zpz<5>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<1>,
          ZPZV<2»; };  // NOLINT
03203     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
03204     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
03205     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
03206     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
03207     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
03208     template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>,
          ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2»; };  // NOLINT
03209     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
03210     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
03211     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<0>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<3»; };  //
          NOLINT
03212     template<> struct ConwayPolynomial<5, 20> { using ZPZ = aerobus::zpz<5>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<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
03213     template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
          POLYV<ZPZV<1>, ZPZV<4»; };  // NOLINT
03214     template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
          POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3»; };  // NOLINT
03215     template<> struct ConwayPolynomial<7, 3> { using ZPZ = aerobus::zpz<7>; using type =
          POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4»; };  // NOLINT
03216     template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; };  // NOLINT
03217     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
03218     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
03219     template<> struct ConwayPolynomial<7, 7> { using ZPZ = aerobus::zpz<7>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<4»; };  // NOLINT
03220     template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<6>, ZPZV<2>, ZPZV<3»; };  // NOLINT
03221     template<> struct ConwayPolynomial<7, 9> { using ZPZ = aerobus::zpz<7>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<4»; };  //
          NOLINT
03222     template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<3>,
          ZPZV<3»; };  // NOLINT
03223     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
03224     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
03225     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
03226     template<> struct ConwayPolynomial<7, 14> { using ZPZ = aerobus::zpz<7>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<6>,
          ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<3»; };  // NOLINT
03227     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<0>, ZPZV<5>,
          ZPZV<6>, ZPZV<6>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<4»; };  // NOLINT
03228     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<0>, ZPZV<4>, ZPZV<5>,
          ZPZV<3>, ZPZV<4>, ZPZV<1>, ZPZV<6>, ZPZV<2>, ZPZV<4>, ZPZV<3»; };  // NOLINT
```

```
03229     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
03230     template<> struct ConwayPolynomial<7, 18> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<6>, ZPZV<1>,
      ZPZV<6>, ZPZV<5>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<2>, ZPZV<3>; };  // NOLINT
03231     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<0>, ZPZV<5>, ZPZV<0>, ZPZV<4>; };  //
      NOLINT
03232     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
03233     template<> struct ConwayPolynomial<11, 1> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<9>; };  // NOLINT
03234     template<> struct ConwayPolynomial<11, 2> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2>; };  // NOLINT
03235     template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<9>; };  // NOLINT
03236     template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2>; };  // NOLINT
03237     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
03238     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
03239     template<> struct ConwayPolynomial<11, 7> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<9>; };  // NOLINT
03240     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
03241     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
03242     template<> struct ConwayPolynomial<11, 10> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<10>, ZPZV<6>, ZPZV<6>,
      ZPZV<2>; };  // NOLINT
03243     template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<10>, ZPZV<9>; };  // NOLINT
03244     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
03245     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<0>, ZPZV<7>, ZPZV<9>; };  // NOLINT
03246     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
03247     template<> struct ConwayPolynomial<11, 15> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>,
      ZPZV<7>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<0>, ZPZV<9>; };  // NOLINT
03248     template<> struct ConwayPolynomial<11, 16> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>,
      ZPZV<1>, ZPZV<3>, ZPZV<5>, ZPZV<3>, ZPZV<10>, ZPZV<9>, ZPZV<2>; };  // NOLINT
03249     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<0>, ZPZV<4>, ZPZV<9>; };  // NOLINT
03250     template<> struct ConwayPolynomial<11, 18> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<10>, ZPZV<8>,
      ZPZV<3>, ZPZV<9>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<9>, ZPZV<8>, ZPZV<2>, ZPZV<2>; };  // NOLINT
03251     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<0>, ZPZV<8>, ZPZV<2>, ZPZV<9>; };  //
      NOLINT
03252     template<> struct ConwayPolynomial<11, 20> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>,
      ZPZV<9>, ZPZV<1>, ZPZV<5>, ZPZV<7>, ZPZV<2>, ZPZV<4>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZPZV<2>; };
      // NOLINT
03253     template<> struct ConwayPolynomial<13, 1> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<11>; };  // NOLINT
03254     template<> struct ConwayPolynomial<13, 2> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<12>, ZPZV<2>; };  // NOLINT
03255     template<> struct ConwayPolynomial<13, 3> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>; };  // NOLINT
03256     template<> struct ConwayPolynomial<13, 4> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2>; };  // NOLINT
03257     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
03258     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
03259     template<> struct ConwayPolynomial<13, 7> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<11>; };  // NOLINT
03260     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
03261     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
```

```
03262    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
03263    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<0>,
      ZPZV<3>, ZPZV<11»; };  // NOLINT
03264    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
03265    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
03266    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<0>, ZPZV<4>, ZPZV<0>, ZPZV<6>,
      ZPZV<11>, ZPZV<7>, ZPZV<10>, ZPZV<10>, ZPZV<2»; };  // NOLINT
03267    template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>,
      ZPZV<2>, ZPZV<11>, ZPZV<10>, ZPZV<11>, ZPZV<8>, ZPZV<11»; };  // NOLINT
03268    template<> struct ConwayPolynomial<13, 16> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<12>,
      ZPZV<8>, ZPZV<2>, ZPZV<12>, ZPZV<9>, ZPZV<12>, ZPZV<6>, ZPZV<2»; };  // NOLINT
03269    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<0>, ZPZV<10>, ZPZV<6>, ZPZV<11»; };  // NOLINT
03270    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<6>, ZPZV<0>, ZPZV<9>, ZPZV<2»; };  // NOLINT
03271    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<0>, ZPZV<9>, ZPZV<11»; };  //
      NOLINT
03272    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
03273    template<> struct ConwayPolynomial<17, 1> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<14»; };  // NOLINT
03274    template<> struct ConwayPolynomial<17, 2> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»; };  // NOLINT
03275    template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14»; };  // NOLINT
03276    template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<10>, ZPZV<3»; };  // NOLINT
03277    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
03278    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
03279    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
03280    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
03281    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<0>, ZPZV<7>, ZPZV<8>, ZPZV<14»; };
      // NOLINT
03282    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
03283    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<0>,
      ZPZV<5>, ZPZV<14»; };  // NOLINT
03284    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
03285    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<0>, ZPZV<15>, ZPZV<14»; };  // NOLINT
03286    template<> struct ConwayPolynomial<17, 14> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<1>, ZPZV<8>,
      ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3»; };  // NOLINT
03287    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<0>, ZPZV<4>,
      ZPZV<4>, ZPZV<16>, ZPZV<6>, ZPZV<14>, ZPZV<14>, ZPZV<14»; };  // NOLINT
03288    template<> struct ConwayPolynomial<17, 16> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<13>,
      ZPZV<5>, ZPZV<2>, ZPZV<12>, ZPZV<13>, ZPZV<12>, ZPZV<1>, ZPZV<3»; };  // NOLINT
03289    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<0>, ZPZV<16>, ZPZV<14»; };  // NOLINT
03290    template<> struct ConwayPolynomial<17, 18> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<16>,
      ZPZV<7>, ZPZV<0>, ZPZV<9>, ZPZV<11>, ZPZV<13>, ZPZV<13>, ZPZV<9>, ZPZV<3»; };  // NOLINT
03291    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<0>, ZPZV<11>, ZPZV<14»; };  //
      NOLINT
03292    template<> struct ConwayPolynomial<17, 20> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<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
03293      template<> struct ConwayPolynomial<19, 1> { using ZPZ = aerobus::zpz<19>; using type =
           POLYV<ZPZV<1>, ZPZV<17»; };  // NOLINT
03294      template<> struct ConwayPolynomial<19, 2> { using ZPZ = aerobus::zpz<19>; using type =
           POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2»; };  // NOLINT
03295      template<> struct ConwayPolynomial<19, 3> { using ZPZ = aerobus::zpz<19>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17»; };  // NOLINT
03296      template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2»; };  // NOLINT
03297      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
03298      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
03299      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
03300      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
03301      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
03302      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
03303      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
03304      template<> struct ConwayPolynomial<19, 12> { using ZPZ = aerobus::zpz<19>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<18>, ZPZV<2>, ZPZV<9>,
           ZPZV<16>, ZPZV<7>, ZPZV<2»; };  // NOLINT
03305      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
03306      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<0>, ZPZV<11>, ZPZV<11>, ZPZV<11>,
           ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<7>, ZPZV<2»; };  // NOLINT
03307      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
03308      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
03309      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
03310      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
03311      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
03312      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
03313      template<> struct ConwayPolynomial<23, 1> { using ZPZ = aerobus::zpz<23>; using type =
           POLYV<ZPZV<1>, ZPZV<18»; };  // NOLINT
03314      template<> struct ConwayPolynomial<23, 2> { using ZPZ = aerobus::zpz<23>; using type =
           POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; };  // NOLINT
03315      template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; };  // NOLINT
03316      template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5»; };  // NOLINT
03317      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
03318      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
03319      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
03320      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
03321      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
03322      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
03323      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<0>, ZPZV<22>,
           ZPZV<7>, ZPZV<18»; };  // NOLINT
03324      template<> struct ConwayPolynomial<23, 12> { using ZPZ = aerobus::zpz<23>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<21>, ZPZV<15>, ZPZV<14>, ZPZV<12>,
           ZPZV<18>, ZPZV<12>, ZPZV<5»; };  // NOLINT
03325      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<0>, ZPZV<9>, ZPZV<18»; };  // NOLINT
```

```
03326      template<> struct ConwayPolynomial<23, 14> { using ZPZ = aerobus::zpz<23>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<1>,
        ZPZV<18>, ZPZV<19>, ZPZV<1>, ZPZV<22>, ZPZV<5»; };  // NOLINT
03327      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<7>, ZPZV<18>, ZPZV<18»; };  // NOLINT
03328      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<0>, ZPZV<0>, ZPZV<19>,
        ZPZV<19>, ZPZV<16>, ZPZV<13>, ZPZV<1>, ZPZV<14>, ZPZV<17>, ZPZV<5»; };  // NOLINT
03329      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<0>, ZPZV<20>, ZPZV<18»; };  // NOLINT
03330      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<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
03331      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<0>, ZPZV<5>, ZPZV<18»; };  //
        NOLINT
03332      template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type =
        POLYV<ZPZV<1>, ZPZV<27»; };  // NOLINT
03333      template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
        POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; };  // NOLINT
03334      template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<27»; };  // NOLINT
03335      template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<2»; };  // NOLINT
03336      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
03337      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
03338      template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<27»; };  // NOLINT
03339      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
03340      template<> struct ConwayPolynomial<29, 9> { using ZPZ = aerobus::zpz<29>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<22>, ZPZV<22>, ZPZV<27»; };
        // NOLINT
03341      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
03342      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<0>, ZPZV<28>,
        ZPZV<8>, ZPZV<27»; };  // NOLINT
03343      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
03344      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<0>, ZPZV<7>, ZPZV<27»; };  // NOLINT
03345      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
03346      template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::zpz<29>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>,
        ZPZV<14>, ZPZV<8>, ZPZV<1>, ZPZV<12>, ZPZV<26>, ZPZV<27»; };  // NOLINT
03347      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<0>, ZPZV<6>, ZPZV<27>,
        ZPZV<2>, ZPZV<18>, ZPZV<23>, ZPZV<1>, ZPZV<27>, ZPZV<10>, ZPZV<2»; };  // NOLINT
03348      template<> struct ConwayPolynomial<29, 17> { using ZPZ = aerobus::zpz<29>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<27»; };  // NOLINT
03349      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
03350      template<> struct ConwayPolynomial<29, 19> { using ZPZ = aerobus::zpz<29>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<27»; };  //
        NOLINT
03351      template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
        POLYV<ZPZV<1>, ZPZV<28»; };  // NOLINT
03352      template<> struct ConwayPolynomial<31, 2> { using ZPZ = aerobus::zpz<31>; using type =
        POLYV<ZPZV<1>, ZPZV<29>, ZPZV<3»; };  // NOLINT
03353      template<> struct ConwayPolynomial<31, 3> { using ZPZ = aerobus::zpz<31>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<28»; };  // NOLINT
03354      template<> struct ConwayPolynomial<31, 4> { using ZPZ = aerobus::zpz<31>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3»; };  // NOLINT
03355      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
03356      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
03357      template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28»; };  // NOLINT
03358      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
```

```
03359    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
03360    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
03361    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<0>,
         ZPZV<20>, ZPZV<28>; };  // NOLINT
03362    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
03363    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<0>, ZPZV<6>, ZPZV<28>; };  // NOLINT
03364    template<> struct ConwayPolynomial<31, 14> { using ZPZ = aerobus::zpz<31>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<5>, ZPZV<1>,
         ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<6>, ZPZV<3>; };  // NOLINT
03365    template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>,
         ZPZV<29>, ZPZV<12>, ZPZV<13>, ZPZV<23>, ZPZV<25>, ZPZV<28>; };  // NOLINT
03366    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<0>, ZPZV<28>,
         ZPZV<24>, ZPZV<26>, ZPZV<28>, ZPZV<11>, ZPZV<19>, ZPZV<27>, ZPZV<3>; };  // NOLINT
03367    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<0>, ZPZV<10>, ZPZV<28>; };  // NOLINT
03368    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
03369    template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28>; };  //
         NOLINT
03370    template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
         POLYV<ZPZV<1>, ZPZV<35>; };  // NOLINT
03371    template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
         POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2>; };  // NOLINT
03372    template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35>; };  // NOLINT
03373    template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2>; };  // NOLINT
03374    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
03375    template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<4>, ZPZV<30>, ZPZV<2>; };  // NOLINT
03376    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
03377    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
03378    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
03379    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
03380    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<0>,
         ZPZV<2>, ZPZV<35>; };  // NOLINT
03381    template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<10>, ZPZV<23>, ZPZV<23>,
         ZPZV<18>, ZPZV<33>, ZPZV<2>; };  // NOLINT
03382    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<0>, ZPZV<6>, ZPZV<35>; };  // NOLINT
03383    template<> struct ConwayPolynomial<37, 14> { using ZPZ = aerobus::zpz<37>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35>, ZPZV<35>, ZPZV<1>,
         ZPZV<32>, ZPZV<16>, ZPZV<1>, ZPZV<9>, ZPZV<2>; };  // NOLINT
03384    template<> struct ConwayPolynomial<37, 15> { using ZPZ = aerobus::zpz<37>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<31>,
         ZPZV<28>, ZPZV<27>, ZPZV<13>, ZPZV<34>, ZPZV<33>, ZPZV<35>; };  // NOLINT
03385    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<0>, ZPZV<3>, ZPZV<35>; };  // NOLINT
03386    template<> struct ConwayPolynomial<37, 18> { using ZPZ = aerobus::zpz<37>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<8>, ZPZV<19>, ZPZV<15>,
         ZPZV<1>, ZPZV<22>, ZPZV<20>, ZPZV<12>, ZPZV<32>, ZPZV<14>, ZPZV<27>, ZPZV<20>, ZPZV<2>; };  // NOLINT
03387    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<0>, ZPZV<36>, ZPZV<23>, ZPZV<35>; };  //
         NOLINT
03388    template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type =
         POLYV<ZPZV<1>, ZPZV<35>; };  // NOLINT
03389    template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
         POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6>; };  // NOLINT
03390    template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<35>; };  // NOLINT
```

```
03391    template<> struct ConwayPolynomial<41, 4> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<6>>; };  // NOLINT
03392    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
03393    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
03394    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
03395    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
03396    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
03397    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
03398    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
03399    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
03400    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
03401    template<> struct ConwayPolynomial<41, 14> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<15>, ZPZV<4>,
    ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6>>; };  // NOLINT
03402    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<0>, ZPZV<29>,
    ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35>>; };  // NOLINT
03403    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<0>, ZPZV<4>, ZPZV<35>>; };  // NOLINT
03404    template<> struct ConwayPolynomial<41, 18> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<20>,
    ZPZV<23>, ZPZV<35>, ZPZV<38>, ZPZV<24>, ZPZV<12>, ZPZV<29>, ZPZV<10>, ZPZV<6>, ZPZV<6>>; };  // NOLINT
03405    template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35>>; };  //
    NOLINT
03406    template<> struct ConwayPolynomial<43, 1> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<40>>; };  // NOLINT
03407    template<> struct ConwayPolynomial<43, 2> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3>>; };  // NOLINT
03408    template<> struct ConwayPolynomial<43, 3> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40>>; };  // NOLINT
03409    template<> struct ConwayPolynomial<43, 4> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3>>; };  // NOLINT
03410    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
03411    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
03412    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
03413    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
03414    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
03415    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
03416    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<0>,
    ZPZV<7>, ZPZV<40>>; };  // NOLINT
03417    template<> struct ConwayPolynomial<43, 12> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<27>, ZPZV<16>, ZPZV<17>, ZPZV<6>,
    ZPZV<23>, ZPZV<38>, ZPZV<3>>; };  // NOLINT
03418    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<0>, ZPZV<4>, ZPZV<40>>; };  // NOLINT
03419    template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<38>, ZPZV<22>, ZPZV<24>,
    ZPZV<37>, ZPZV<18>, ZPZV<4>, ZPZV<19>, ZPZV<3>>; };  // NOLINT
03420    template<> struct ConwayPolynomial<43, 15> { using ZPZ = aerobus::zpz<43>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<37>,
    ZPZV<22>, ZPZV<42>, ZPZV<4>, ZPZV<15>, ZPZV<37>, ZPZV<40>>; };  // NOLINT
03421    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<0>, ZPZV<36>, ZPZV<40>>; };  // NOLINT
03422    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
03423    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<0>, ZPZV<30>, ZPZV<40»; };  //
       NOLINT
03424     template<> struct ConwayPolynomial<47, 1> { using ZPZ = aerobus::zpz<47>; using type =
       POLYV<ZPZV<1>, ZPZV<42»; };  // NOLINT
03425     template<> struct ConwayPolynomial<47, 2> { using ZPZ = aerobus::zpz<47>; using type =
       POLYV<ZPZV<1>, ZPZV<45>, ZPZV<5»; };  // NOLINT
03426     template<> struct ConwayPolynomial<47, 3> { using ZPZ = aerobus::zpz<47>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42»; };  // NOLINT
03427     template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; };  // NOLINT
03428     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
03429     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
03430     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
03431     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
03432     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
03433     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
03434     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<0>,
       ZPZV<6>, ZPZV<42»; };  // NOLINT
03435     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
03436     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
03437     template<> struct ConwayPolynomial<47, 14> { using ZPZ = aerobus::zpz<47>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<20>, ZPZV<30>,
       ZPZV<17>, ZPZV<24>, ZPZV<9>, ZPZV<32>, ZPZV<5»; };  // NOLINT
03438     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<0>, ZPZV<43>,
       ZPZV<31>, ZPZV<14>, ZPZV<42>, ZPZV<13>, ZPZV<17>, ZPZV<42»; };  // NOLINT
03439     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<0>, ZPZV<16>, ZPZV<42»; };  // NOLINT
03440     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
03441     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<0>, ZPZV<35>, ZPZV<42»; };  //
       NOLINT
03442     template<> struct ConwayPolynomial<53, 1> { using ZPZ = aerobus::zpz<53>; using type =
       POLYV<ZPZV<1>, ZPZV<51»; };  // NOLINT
03443     template<> struct ConwayPolynomial<53, 2> { using ZPZ = aerobus::zpz<53>; using type =
       POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2»; };  // NOLINT
03444     template<> struct ConwayPolynomial<53, 3> { using ZPZ = aerobus::zpz<53>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51»; };  // NOLINT
03445     template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<38>, ZPZV<2»; };  // NOLINT
03446     template<> struct ConwayPolynomial<53, 5> { using ZPZ = aerobus::zpz<53>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<51»; };  // NOLINT
03447     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
03448     template<> struct ConwayPolynomial<53, 7> { using ZPZ = aerobus::zpz<53>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<51»; };  // NOLINT
03449     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
03450     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
03451     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
03452     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<0>,
       ZPZV<15>, ZPZV<51»; };  // NOLINT
03453     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
03454     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<0>, ZPZV<52>, ZPZV<28>, ZPZV<51»; };  // NOLINT
03455     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
03456     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<0>, ZPZV<22>,
       ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51»; };  // NOLINT
03457     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<0>, ZPZV<12>, ZPZV<51»; };  // NOLINT
03458    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
03459    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<0>, ZPZV<11>, ZPZV<51»; };  //
       NOLINT
03460    template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
       POLYV<ZPZV<1>, ZPZV<57»; };  // NOLINT
03461    template<> struct ConwayPolynomial<59, 2> { using ZPZ = aerobus::zpz<59>; using type =
       POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2»; };  // NOLINT
03462    template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; };  // NOLINT
03463    template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<2»; };  // NOLINT
03464    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
03465    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
03466    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
03467    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
03468    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
03469    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
03470    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
03471    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
03472    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
03473    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
03474    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<0>, ZPZV<57>,
       ZPZV<24>, ZPZV<23>, ZPZV<13>, ZPZV<39>, ZPZV<58>, ZPZV<57»; };  // NOLINT
03475    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<0>, ZPZV<9>, ZPZV<57»; };  // NOLINT
03476    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<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
03477    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<0>, ZPZV<11>, ZPZV<57»; };  //
       NOLINT
03478    template<> struct ConwayPolynomial<61, 1> { using ZPZ = aerobus::zpz<61>; using type =
       POLYV<ZPZV<1>, ZPZV<59»; };  // NOLINT
03479    template<> struct ConwayPolynomial<61, 2> { using ZPZ = aerobus::zpz<61>; using type =
       POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; };  // NOLINT
03480    template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; };  // NOLINT
03481    template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2»; };  // NOLINT
03482    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
03483    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
03484    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
03485    template<> struct ConwayPolynomial<61, 8> { using ZPZ = aerobus::zpz<61>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<1>, ZPZV<56>, ZPZV<2»; };  // NOLINT
03486    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
03487    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
03488    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<0>,
       ZPZV<18>, ZPZV<59»; };  // NOLINT
03489    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
03490    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<0>, ZPZV<3>, ZPZV<59»; };  // NOLINT
03491    template<> struct ConwayPolynomial<61, 14> { using ZPZ = aerobus::zpz<61>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<48>, ZPZV<26>, ZPZV<11>,
        ZPZV<8>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<2»; };  // NOLINT
03492    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<0>, ZPZV<39>,
        ZPZV<35>, ZPZV<44>, ZPZV<25>, ZPZV<23>, ZPZV<51>, ZPZV<59»; };  // NOLINT
03493    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<0>, ZPZV<10>, ZPZV<59»; };  // NOLINT
03494    template<> struct ConwayPolynomial<61, 18> { using ZPZ = aerobus::zpz<61>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35>, ZPZV<36>, ZPZV<13>,
        ZPZV<36>, ZPZV<4>, ZPZV<32>, ZPZV<57>, ZPZV<42>, ZPZV<25>, ZPZV<25>, ZPZV<52>, ZPZV<2»; };  // NOLINT
03495    template<> struct ConwayPolynomial<61, 19> { using ZPZ = aerobus::zpz<61>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; };  //
        NOLINT
03496    template<> struct ConwayPolynomial<67, 1> { using ZPZ = aerobus::zpz<67>; using type =
        POLYV<ZPZV<1>, ZPZV<65»; };  // NOLINT
03497    template<> struct ConwayPolynomial<67, 2> { using ZPZ = aerobus::zpz<67>; using type =
        POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; };  // NOLINT
03498    template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65»; };  // NOLINT
03499    template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<54>, ZPZV<2»; };  // NOLINT
03500    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
03501    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
03502    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
03503    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
03504    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
03505    template<> struct ConwayPolynomial<67, 10> { using ZPZ = aerobus::zpz<67>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<0>, ZPZV<16>, ZPZV<7>, ZPZV<23>,
        ZPZV<2»; };  // NOLINT
03506    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<0>, ZPZV<66>,
        ZPZV<9>, ZPZV<65»; };  // NOLINT
03507    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
03508    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<0>, ZPZV<22>, ZPZV<65»; };  // NOLINT
03509    template<> struct ConwayPolynomial<67, 14> { using ZPZ = aerobus::zpz<67>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<22>, ZPZV<5>,
        ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<2»; };  // NOLINT
03510    template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>,
        ZPZV<52>, ZPZV<41>, ZPZV<20>, ZPZV<21>, ZPZV<46>, ZPZV<65»; };  // NOLINT
03511    template<> struct ConwayPolynomial<67, 17> { using ZPZ = aerobus::zpz<67>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<65»; };  // NOLINT
03512    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
03513    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<0>, ZPZV<18>, ZPZV<65»; };  //
        NOLINT
03514    template<> struct ConwayPolynomial<71, 1> { using ZPZ = aerobus::zpz<71>; using type =
        POLYV<ZPZV<1>, ZPZV<64»; };  // NOLINT
03515    template<> struct ConwayPolynomial<71, 2> { using ZPZ = aerobus::zpz<71>; using type =
        POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7»; };  // NOLINT
03516    template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; };  // NOLINT
03517    template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, ZPZV<7»; };  // NOLINT
03518    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
03519    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
03520    template<> struct ConwayPolynomial<71, 7> { using ZPZ = aerobus::zpz<71>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<64»; };  // NOLINT
03521    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
03522    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
03523    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
03524    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<0>,
      ZPZV<48>, ZPZV<64»; };  // NOLINT
03525    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
03526    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<0>, ZPZV<27>, ZPZV<64»; };  // NOLINT
03527    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<0>, ZPZV<28>,
      ZPZV<32>, ZPZV<18>, ZPZV<52>, ZPZV<67>, ZPZV<49>, ZPZV<64»; };  // NOLINT
03528    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<0>, ZPZV<0>, ZPZV<3>, ZPZV<64»; };  // NOLINT
03529    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
03530    template<> struct ConwayPolynomial<73, 1> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<68»; };  // NOLINT
03531    template<> struct ConwayPolynomial<73, 2> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5»; };  // NOLINT
03532    template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68»; };  // NOLINT
03533    template<> struct ConwayPolynomial<73, 4> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5»; };  // NOLINT
03534    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
03535    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
03536    template<> struct ConwayPolynomial<73, 7> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<68»; };  // NOLINT
03537    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
03538    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
03539    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
03540    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<0>,
      ZPZV<5>, ZPZV<68»; };  // NOLINT
03541    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
03542    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<0>, ZPZV<7>, ZPZV<68»; };  // NOLINT
03543    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<0>,
      ZPZV<10>, ZPZV<33>, ZPZV<57>, ZPZV<57>, ZPZV<62>, ZPZV<68»; };  // NOLINT
03544    template<> struct ConwayPolynomial<73, 17> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<68»; };  // NOLINT
03545    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
03546    template<> struct ConwayPolynomial<79, 1> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<76»; };  // NOLINT
03547    template<> struct ConwayPolynomial<79, 2> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3»; };  // NOLINT
03548    template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76»; };  // NOLINT
03549    template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3»; };  // NOLINT
03550    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
03551    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
03552    template<> struct ConwayPolynomial<79, 7> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76»; };  // NOLINT
03553    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
03554    template<> struct ConwayPolynomial<79, 9> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<19>, ZPZV<76»; };
      // NOLINT
03555    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
03556    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<0>,
```

```
      ZPZV<3>, ZPZV<76»; };  // NOLINT
03557     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
03558     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<0>, ZPZV<78>, ZPZV<4>, ZPZV<76»; };  // NOLINT
03559     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<0>, ZPZV<25>, ZPZV<76»; };  // NOLINT
03560     template<> struct ConwayPolynomial<79, 19> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<76»; };  //
      NOLINT
03561     template<> struct ConwayPolynomial<83, 1> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<81»; };  // NOLINT
03562     template<> struct ConwayPolynomial<83, 2> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; };  // NOLINT
03563     template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81»; };  // NOLINT
03564     template<> struct ConwayPolynomial<83, 4> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2»; };  // NOLINT
03565     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
03566     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
03567     template<> struct ConwayPolynomial<83, 7> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<81»; };  // NOLINT
03568     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
03569     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
03570     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
03571     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<0>,
      ZPZV<17>, ZPZV<81»; };  // NOLINT
03572     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
03573     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<0>, ZPZV<15>, ZPZV<81»; };  // NOLINT
03574     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<0>, ZPZV<7>, ZPZV<81»; };  // NOLINT
03575     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<0>, ZPZV<47>, ZPZV<81»; };  //
      NOLINT
03576     template<> struct ConwayPolynomial<89, 1> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<86»; };  // NOLINT
03577     template<> struct ConwayPolynomial<89, 2> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; };  // NOLINT
03578     template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<86»; };  // NOLINT
03579     template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; };  // NOLINT
03580     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
03581     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
03582     template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<86»; };  // NOLINT
03583     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
03584     template<> struct ConwayPolynomial<89, 9> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<12>, ZPZV<6>, ZPZV<86»; };
      // NOLINT
03585     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
03586     template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<88>,
      ZPZV<26>, ZPZV<86»; };  // NOLINT
03587     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
03588     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<0>, ZPZV<17>, ZPZV<86»; };  // NOLINT
03589     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
03590     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<34>, ZPZV<86»; };  //
      NOLINT
03591     template<> struct ConwayPolynomial<97, 1> { using ZPZ = aerobus::zpz<97>; using type =
      POLYV<ZPZV<1>, ZPZV<92»; };  // NOLINT
03592     template<> struct ConwayPolynomial<97, 2> { using ZPZ = aerobus::zpz<97>; using type =
      POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5»; };  // NOLINT
03593     template<> struct ConwayPolynomial<97, 3> { using ZPZ = aerobus::zpz<97>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<92»; };  // NOLINT
03594     template<> struct ConwayPolynomial<97, 4> { using ZPZ = aerobus::zpz<97>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; };  // NOLINT
03595     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
03596     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
03597     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
03598     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
03599     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
03600     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
03601     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
03602     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
03603     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
03604     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
03605     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
03606     template<> struct ConwayPolynomial<101, 1> { using ZPZ = aerobus::zpz<101>; using type =
      POLYV<ZPZV<1>, ZPZV<99»; };  // NOLINT
03607     template<> struct ConwayPolynomial<101, 2> { using ZPZ = aerobus::zpz<101>; using type =
      POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; };  // NOLINT
03608     template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<99»; };  // NOLINT
03609     template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<78>, ZPZV<2»; };  // NOLINT
03610     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
03611     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
03612     template<> struct ConwayPolynomial<101, 7> { using ZPZ = aerobus::zpz<101>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<99»; };  // NOLINT
03613     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
03614     template<> struct ConwayPolynomial<101, 9> { using ZPZ = aerobus::zpz<101>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<64>, ZPZV<47>, ZPZV<99»; };
      // NOLINT
03615     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
03616     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
03617     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
03618     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
03619     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
03620     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
03621     template<> struct ConwayPolynomial<103, 1> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<98»; };  // NOLINT
03622     template<> struct ConwayPolynomial<103, 2> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; };  // NOLINT
03623     template<> struct ConwayPolynomial<103, 3> { using ZPZ = aerobus::zpz<103>; using type =
```

```
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<98»; };  // NOLINT
03624      template<> struct ConwayPolynomial<103, 4> { using ZPZ = aerobus::zpz<103>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<88>, ZPZV<5»; };  // NOLINT
03625      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
03626      template<> struct ConwayPolynomial<103, 6> { using ZPZ = aerobus::zpz<103>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<9>, ZPZV<30>, ZPZV<5»; };  // NOLINT
03627      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
03628      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
03629      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
03630      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
03631      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
03632      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
03633      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
03634      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
03635      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
03636      template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
          POLYV<ZPZV<1>, ZPZV<105»; };  // NOLINT
03637      template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
          POLYV<ZPZV<1>, ZPZV<103>, ZPZV<2»; };  // NOLINT
03638      template<> struct ConwayPolynomial<107, 3> { using ZPZ = aerobus::zpz<107>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<105»; };  // NOLINT
03639      template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; };  // NOLINT
03640      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
03641      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
03642      template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<105»; };  // NOLINT
03643      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
03644      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
03645      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
03646      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
03647      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
03648      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<4>, ZPZV<105»; };  // NOLINT
03649      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<2>, ZPZV<105»; };  // NOLINT
03650      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
03651      template<> struct ConwayPolynomial<109, 1> { using ZPZ = aerobus::zpz<109>; using type =
          POLYV<ZPZV<1>, ZPZV<103»; };  // NOLINT
03652      template<> struct ConwayPolynomial<109, 2> { using ZPZ = aerobus::zpz<109>; using type =
          POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; };  // NOLINT
03653      template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103»; };  // NOLINT
03654      template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; };  // NOLINT
03655      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
03656      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
03657      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
```

```
03658      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
03659      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
03660      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
03661      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<11>, ZPZV<103»; }; // NOLINT
03662      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
03663      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
03664      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
03665      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<15>, ZPZV<103»; }; //
      NOLINT
03666      template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<110»; }; // NOLINT
03667      template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3»; }; // NOLINT
03668      template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<110»; }; // NOLINT
03669      template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<3»; }; // NOLINT
03670      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
03671      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
03672      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
03673      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
03674      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
03675      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
03676      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
03677      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
03678      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
03679      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
03680      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
03681      template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<124»; }; // NOLINT
03682      template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3»; }; // NOLINT
03683      template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124»; }; // NOLINT
03684      template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<97>, ZPZV<3»; }; // NOLINT
03685      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
03686      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
03687      template<> struct ConwayPolynomial<127, 7> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<124»; }; // NOLINT
03688      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
03689      template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<119>, ZPZV<126>, ZPZV<124»;
      }; // NOLINT
03690      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
```

```
03691    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
03692    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
03693    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<0>, ZPZV<8>, ZPZV<124>; };  // NOLINT
03694    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<0>, ZPZV<9>, ZPZV<124>; };  // NOLINT
03695    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
03696    template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
         POLYV<ZPZV<1>, ZPZV<129>; };  // NOLINT
03697    template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
         POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2>; };  // NOLINT
03698    template<> struct ConwayPolynomial<131, 3> { using ZPZ = aerobus::zpz<131>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<129>; };  // NOLINT
03699    template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2>; };  // NOLINT
03700    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
03701    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
03702    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
03703    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
03704    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
03705    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
03706    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<0>,
         ZPZV<6>, ZPZV<129>; };  // NOLINT
03707    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
03708    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<0>, ZPZV<9>, ZPZV<129>; };  // NOLINT
03709    template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<129>; };  // NOLINT
03710    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<0>, ZPZV<9>, ZPZV<129>; };  //
         NOLINT
03711    template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
         POLYV<ZPZV<1>, ZPZV<134>; };  // NOLINT
03712    template<> struct ConwayPolynomial<137, 2> { using ZPZ = aerobus::zpz<137>; using type =
         POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3>; };  // NOLINT
03713    template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134>; };  // NOLINT
03714    template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<95>, ZPZV<3>; };  // NOLINT
03715    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
03716    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
03717    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
03718    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
03719    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
03720    template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<20>, ZPZV<67>, ZPZV<93>, ZPZV<119>,
         ZPZV<3>; };  // NOLINT
03721    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<0>,
         ZPZV<1>, ZPZV<134>; };  // NOLINT
03722    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
03723    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
```

```
03724     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<0>, ZPZV<136>, ZPZV<4>, ZPZV<134»; };   // NOLINT
03725     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<0>, ZPZV<18>, ZPZV<134»; };   //
      NOLINT
03726     template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
      POLYV<ZPZV<1>, ZPZV<137»; };   // NOLINT
03727     template<> struct ConwayPolynomial<139, 2> { using ZPZ = aerobus::zpz<139>; using type =
      POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2»; };   // NOLINT
03728     template<> struct ConwayPolynomial<139, 3> { using ZPZ = aerobus::zpz<139>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<137»; };   // NOLINT
03729     template<> struct ConwayPolynomial<139, 4> { using ZPZ = aerobus::zpz<139>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2»; };   // NOLINT
03730     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
03731     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
03732     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
03733     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
03734     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
03735     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
03736     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
03737     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
03738     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<0>, ZPZV<22>, ZPZV<137»; };   // NOLINT
03739     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<0>, ZPZV<2>, ZPZV<137»; };   // NOLINT
03740     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<0>, ZPZV<23>, ZPZV<137»; };   //
      NOLINT
03741     template<> struct ConwayPolynomial<149, 1> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<147»; };   // NOLINT
03742     template<> struct ConwayPolynomial<149, 2> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<145>, ZPZV<2»; };   // NOLINT
03743     template<> struct ConwayPolynomial<149, 3> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147»; };   // NOLINT
03744     template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<2»; };   // NOLINT
03745     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
03746     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
03747     template<> struct ConwayPolynomial<149, 7> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<147»; };   // NOLINT
03748     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
03749     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
03750     template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<74>, ZPZV<42>, ZPZV<148>, ZPZV<143>, ZPZV<51>,
      ZPZV<2»; };   // NOLINT
03751     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<0>,
      ZPZV<33>, ZPZV<147»; };   // NOLINT
03752     template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<91>, ZPZV<52>, ZPZV<9>,
      ZPZV<104>, ZPZV<110>, ZPZV<2»; };   // NOLINT
03753     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<0>, ZPZV<4>, ZPZV<147»; };   // NOLINT
03754     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<0>, ZPZV<29>, ZPZV<147»; };   // NOLINT
03755     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<0>, ZPZV<5>, ZPZV<147»; };   //
      NOLINT
03756     template<> struct ConwayPolynomial<151, 1> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<145»; };   // NOLINT
```

```
03757     template<> struct ConwayPolynomial<151, 2> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6»; };  // NOLINT
03758     template<> struct ConwayPolynomial<151, 3> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<145»; };  // NOLINT
03759     template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6»; };  // NOLINT
03760     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
03761     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
03762     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
03763     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
03764     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
03765     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
03766     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<0>,
      ZPZV<1>, ZPZV<145»; };  // NOLINT
03767     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
03768     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<0>, ZPZV<12>, ZPZV<145»; };  // NOLINT
03769     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<0>, ZPZV<0>, ZPZV<24>, ZPZV<145»; };  // NOLINT
03770     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<0>, ZPZV<9>, ZPZV<145»; };  //
      NOLINT
03771     template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<152»; };  // NOLINT
03772     template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5»; };  // NOLINT
03773     template<> struct ConwayPolynomial<157, 3> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<152»; };  // NOLINT
03774     template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<136>, ZPZV<5»; };  // NOLINT
03775     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
03776     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
03777     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
03778     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
03779     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
03780     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
03781     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<0>,
      ZPZV<29>, ZPZV<152»; };  // NOLINT
03782     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
03783     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<0>, ZPZV<156>, ZPZV<9>, ZPZV<152»; };  // NOLINT
03784     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<0>, ZPZV<12>, ZPZV<152»; };  // NOLINT
03785     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<0>, ZPZV<14>, ZPZV<152»; };  //
      NOLINT
03786     template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
      POLYV<ZPZV<1>, ZPZV<161»; };  // NOLINT
03787     template<> struct ConwayPolynomial<163, 2> { using ZPZ = aerobus::zpz<163>; using type =
      POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; };  // NOLINT
03788     template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<161»; };  // NOLINT
03789     template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2»; };  // NOLINT
03790     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
03791     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
03792     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
03793     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
03794     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
03795     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
03796     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
03797     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
03798     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
03799     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
03800     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<8>, ZPZV<161»; };  //
          NOLINT
03801     template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
          POLYV<ZPZV<1>, ZPZV<162»; };  // NOLINT
03802     template<> struct ConwayPolynomial<167, 2> { using ZPZ = aerobus::zpz<167>; using type =
          POLYV<ZPZV<1>, ZPZV<166>, ZPZV<5»; };  // NOLINT
03803     template<> struct ConwayPolynomial<167, 3> { using ZPZ = aerobus::zpz<167>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162»; };  // NOLINT
03804     template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; };  // NOLINT
03805     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
03806     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
03807     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
03808     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
03809     template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<122>, ZPZV<162»;
          };  // NOLINT
03810     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
03811     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
03812     template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<142>, ZPZV<10>, ZPZV<142>, ZPZV<131>,
          ZPZV<140>, ZPZV<41>, ZPZV<57>, ZPZV<5»; };  // NOLINT
03813     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
03814     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
03815     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
03816     template<> struct ConwayPolynomial<173, 1> { using ZPZ = aerobus::zpz<173>; using type =
          POLYV<ZPZV<1>, ZPZV<171»; };  // NOLINT
03817     template<> struct ConwayPolynomial<173, 2> { using ZPZ = aerobus::zpz<173>; using type =
          POLYV<ZPZV<1>, ZPZV<169>, ZPZV<2»; };  // NOLINT
03818     template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171»; };  // NOLINT
03819     template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<2»; };  // NOLINT
03820     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
03821     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
03822     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
03823     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
03824     template<> struct ConwayPolynomial<173, 9> { using ZPZ = aerobus::zpz<173>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<104>, ZPZV<171»;
          };  // NOLINT
```

```
03825    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
03826    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
03827    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
03828    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<0>, ZPZV<6>, ZPZV<171»; };  // NOLINT
03829    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<7>, ZPZV<171»; };  // NOLINT
03830    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
03831    template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
     POLYV<ZPZV<1>, ZPZV<177»; };  // NOLINT
03832    template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
     POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2»; };  // NOLINT
03833    template<> struct ConwayPolynomial<179, 3> { using ZPZ = aerobus::zpz<179>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<177»; };  // NOLINT
03834    template<> struct ConwayPolynomial<179, 4> { using ZPZ = aerobus::zpz<179>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<2»; };  // NOLINT
03835    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
03836    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
03837    template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<177»; };  // NOLINT
03838    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
03839    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
03840    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
03841    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
03842    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
03843    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
03844    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
03845    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<11>, ZPZV<177»; };  //
     NOLINT
03846    template<> struct ConwayPolynomial<181, 1> { using ZPZ = aerobus::zpz<181>; using type =
     POLYV<ZPZV<1>, ZPZV<179»; };  // NOLINT
03847    template<> struct ConwayPolynomial<181, 2> { using ZPZ = aerobus::zpz<181>; using type =
     POLYV<ZPZV<1>, ZPZV<177>, ZPZV<2»; };  // NOLINT
03848    template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179»; };  // NOLINT
03849    template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<105>, ZPZV<2»; };  // NOLINT
03850    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
03851    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
03852    template<> struct ConwayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<179»; };  // NOLINT
03853    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
03854    template<> struct ConwayPolynomial<181, 9> { using ZPZ = aerobus::zpz<181>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<107>, ZPZV<168>, ZPZV<179»;
     };  // NOLINT
03855    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
03856    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<0>,
     ZPZV<24>, ZPZV<179»; };  // NOLINT
03857    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
```

```
03858    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<0>, ZPZV<0>, ZPZV<8>, ZPZV<179»; };  // NOLINT
03859    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<0>, ZPZV<0>, ZPZV<9>, ZPZV<179»; };  // NOLINT
03860    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<0>, ZPZV<36>, ZPZV<179»; };  //
    NOLINT
03861    template<> struct ConwayPolynomial<191, 1> { using ZPZ = aerobus::zpz<191>; using type =
    POLYV<ZPZV<1>, ZPZV<172»; };  // NOLINT
03862    template<> struct ConwayPolynomial<191, 2> { using ZPZ = aerobus::zpz<191>; using type =
    POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19»; };  // NOLINT
03863    template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172»; };  // NOLINT
03864    template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; };  // NOLINT
03865    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
03866    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
03867    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
03868    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
03869    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
03870    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
03871    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<0>,
    ZPZV<6>, ZPZV<172»; };  // NOLINT
03872    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
03873    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<0>, ZPZV<12>, ZPZV<172»; };  // NOLINT
03874    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<0>, ZPZV<2>, ZPZV<172»; };  // NOLINT
03875    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
03876    template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
    POLYV<ZPZV<1>, ZPZV<188»; };  // NOLINT
03877    template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
    POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5»; };  // NOLINT
03878    template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; };  // NOLINT
03879    template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<5»; };  // NOLINT
03880    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
03881    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
03882    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
03883    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
03884    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
03885    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
03886    template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<1>, ZPZV<188»; };  // NOLINT
03887    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
03888    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<0>, ZPZV<39>, ZPZV<188»; };  // NOLINT
03889    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<0>, ZPZV<9>, ZPZV<188»; };  // NOLINT
03890    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
03891     template<> struct ConwayPolynomial<197, 1> { using ZPZ = aerobus::zpz<197>; using type =
        POLYV<ZPZV<1>, ZPZV<195»; };   // NOLINT
03892     template<> struct ConwayPolynomial<197, 2> { using ZPZ = aerobus::zpz<197>; using type =
        POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2»; };   // NOLINT
03893     template<> struct ConwayPolynomial<197, 3> { using ZPZ = aerobus::zpz<197>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<195»; };   // NOLINT
03894     template<> struct ConwayPolynomial<197, 4> { using ZPZ = aerobus::zpz<197>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<124>, ZPZV<2»; };   // NOLINT
03895     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
03896     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
03897     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
03898     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
03899     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
03900     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
03901     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<0>,
        ZPZV<14>, ZPZV<195»; };   // NOLINT
03902     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
03903     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
03904     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
03905     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<6>, ZPZV<195»; };   //
        NOLINT
03906     template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
        POLYV<ZPZV<1>, ZPZV<196»; };   // NOLINT
03907     template<> struct ConwayPolynomial<199, 2> { using ZPZ = aerobus::zpz<199>; using type =
        POLYV<ZPZV<1>, ZPZV<193>, ZPZV<3»; };   // NOLINT
03908     template<> struct ConwayPolynomial<199, 3> { using ZPZ = aerobus::zpz<199>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; };   // NOLINT
03909     template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; };   // NOLINT
03910     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
03911     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
03912     template<> struct ConwayPolynomial<199, 7> { using ZPZ = aerobus::zpz<199>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; };   // NOLINT
03913     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
03914     template<> struct ConwayPolynomial<199, 9> { using ZPZ = aerobus::zpz<199>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<177>, ZPZV<141>, ZPZV<196»;
        };   // NOLINT
03915     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
03916     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<0>,
        ZPZV<1>, ZPZV<196»; };   // NOLINT
03917     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
03918     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<0>, ZPZV<10>, ZPZV<196»; };   // NOLINT
03919     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<0>, ZPZV<13>, ZPZV<196»; };   // NOLINT
03920     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
03921     template<> struct ConwayPolynomial<211, 1> { using ZPZ = aerobus::zpz<211>; using type =
        POLYV<ZPZV<1>, ZPZV<209»; };   // NOLINT
03922     template<> struct ConwayPolynomial<211, 2> { using ZPZ = aerobus::zpz<211>; using type =
        POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; };   // NOLINT
03923     template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<209»; };   // NOLINT
03924     template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2»; };   // NOLINT
```

```
03925     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
03926     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
03927     template<> struct ConwayPolynomial<211, 7> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<209»; };  // NOLINT
03928     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
03929     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
03930     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
03931     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
03932     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
03933     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<0>, ZPZV<12>, ZPZV<209»; };  // NOLINT
03934     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
03935     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
03936     template<> struct ConwayPolynomial<223, 1> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<220»; };  // NOLINT
03937     template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; };  // NOLINT
03938     template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<220»; };  // NOLINT
03939     template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<3»; };  // NOLINT
03940     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
03941     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
03942     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
03943     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
03944     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
03945     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
03946     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
03947     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
03948     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
03949     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
03950     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
03951     template<> struct ConwayPolynomial<227, 1> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<225»; };  // NOLINT
03952     template<> struct ConwayPolynomial<227, 2> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2»; };  // NOLINT
03953     template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<225»; };  // NOLINT
03954     template<> struct ConwayPolynomial<227, 4> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2»; };  // NOLINT
03955     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
03956     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
03957     template<> struct ConwayPolynomial<227, 7> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<225»; };  // NOLINT
03958     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
```

```
03959      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
03960      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
03961      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<0>,
      ZPZV<2>, ZPZV<225»; }; // NOLINT
03962      template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<123>, ZPZV<99>, ZPZV<160>, ZPZV<96>,
      ZPZV<127>, ZPZV<142>, ZPZV<94>, ZPZV<2»; }; // NOLINT
03963      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
03964      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<0>, ZPZV<8>, ZPZV<225»; }; // NOLINT
03965      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<0>, ZPZV<34>, ZPZV<225»; }; //
      NOLINT
03966      template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<223»; }; // NOLINT
03967      template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»; }; // NOLINT
03968      template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; }; // NOLINT
03969      template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<6»; }; // NOLINT
03970      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
03971      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
03972      template<> struct ConwayPolynomial<229, 7> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<223»; }; // NOLINT
03973      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
03974      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
03975      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
03976      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<0>,
      ZPZV<2>, ZPZV<223»; }; // NOLINT
03977      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
03978      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<0>, ZPZV<47>, ZPZV<223»; }; // NOLINT
03979      template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<223»; }; // NOLINT
03980      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<0>, ZPZV<228>, ZPZV<15>, ZPZV<223»; }; //
      NOLINT
03981      template<> struct ConwayPolynomial<233, 1> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<230»; }; // NOLINT
03982      template<> struct ConwayPolynomial<233, 2> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
03983      template<> struct ConwayPolynomial<233, 3> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<230»; }; // NOLINT
03984      template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<3»; }; // NOLINT
03985      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
03986      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
03987      template<> struct ConwayPolynomial<233, 7> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230»; }; // NOLINT
03988      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
03989      template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<56>, ZPZV<146>, ZPZV<230»;
      }; // NOLINT
03990      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
03991      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
```

```
03992    template<> struct ConwayPolynomial<233, 12> { using ZPZ = aerobus::zpz<233>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<96>, ZPZV<21>, ZPZV<114>, ZPZV<31>, ZPZV<19>,
         ZPZV<216>, ZPZV<20>, ZPZV<3>; }; // NOLINT
03993    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<0>, ZPZV<14>, ZPZV<230>; }; // NOLINT
03994    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<0>, ZPZV<4>, ZPZV<230>; }; // NOLINT
03995    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<0>, ZPZV<25>, ZPZV<230>; }; //
         NOLINT
03996    template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
         POLYV<ZPZV<1>, ZPZV<232>; }; // NOLINT
03997    template<> struct ConwayPolynomial<239, 2> { using ZPZ = aerobus::zpz<239>; using type =
         POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7>; }; // NOLINT
03998    template<> struct ConwayPolynomial<239, 3> { using ZPZ = aerobus::zpz<239>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<232>; }; // NOLINT
03999    template<> struct ConwayPolynomial<239, 4> { using ZPZ = aerobus::zpz<239>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7>; }; // NOLINT
04000    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
04001    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
04002    template<> struct ConwayPolynomial<239, 7> { using ZPZ = aerobus::zpz<239>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<232>; }; // NOLINT
04003    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
04004    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
04005    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
04006    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<0>,
         ZPZV<8>, ZPZV<232>; }; // NOLINT
04007    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
04008    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<0>, ZPZV<2>, ZPZV<232>; }; // NOLINT
04009    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<0>, ZPZV<9>, ZPZV<232>; }; // NOLINT
04010    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<0>, ZPZV<24>, ZPZV<232>; }; //
         NOLINT
04011    template<> struct ConwayPolynomial<241, 1> { using ZPZ = aerobus::zpz<241>; using type =
         POLYV<ZPZV<1>, ZPZV<234>; }; // NOLINT
04012    template<> struct ConwayPolynomial<241, 2> { using ZPZ = aerobus::zpz<241>; using type =
         POLYV<ZPZV<1>, ZPZV<238>, ZPZV<7>; }; // NOLINT
04013    template<> struct ConwayPolynomial<241, 3> { using ZPZ = aerobus::zpz<241>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<234>; }; // NOLINT
04014    template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<152>, ZPZV<7>; }; // NOLINT
04015    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
04016    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
04017    template<> struct ConwayPolynomial<241, 7> { using ZPZ = aerobus::zpz<241>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<234>; }; // NOLINT
04018    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
04019    template<> struct ConwayPolynomial<241, 9> { using ZPZ = aerobus::zpz<241>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<236>, ZPZV<125>, ZPZV<234>;
         }; // NOLINT
04020    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
04021    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<0>,
         ZPZV<3>, ZPZV<234>; }; // NOLINT
04022    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
04023    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<0>, ZPZV<11>, ZPZV<234>; }; // NOLINT
04024    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
```

```
04025      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
04026      template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<245>; };  // NOLINT
04027      template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6>; };  // NOLINT
04028      template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<245>; };  // NOLINT
04029      template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<200>, ZPZV<6>; };  // NOLINT
04030      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
04031      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
04032      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
04033      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
04034      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
04035      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
04036      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
04037      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
04038      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
04039      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
04040      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
04041      template<> struct ConwayPolynomial<257, 1> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<254>; };  // NOLINT
04042      template<> struct ConwayPolynomial<257, 2> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3>; };  // NOLINT
04043      template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<254>; };  // NOLINT
04044      template<> struct ConwayPolynomial<257, 4> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<187>, ZPZV<3>; };  // NOLINT
04045      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
04046      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
04047      template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<254>; };  // NOLINT
04048      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
04049      template<> struct ConwayPolynomial<257, 9> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<50>, ZPZV<254>;
      };  // NOLINT
04050      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
04051      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
04052      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
04053      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
04054      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
04055      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
04056      template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<258>; };  // NOLINT
04057      template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5>; };  // NOLINT
04058      template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
```

```
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; };  // NOLINT
04059     template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; };  // NOLINT
04060     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
04061     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
04062     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
04063     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
04064     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
04065     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
04066     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
04067     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
04068     template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
          POLYV<ZPZV<1>, ZPZV<267»; };  // NOLINT
04069     template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
          POLYV<ZPZV<1>, ZPZV<268>, ZPZV<2»; };  // NOLINT
04070     template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<267»; };  // NOLINT
04071     template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2»; };  // NOLINT
04072     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
04073     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
04074     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
04075     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
04076     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
04077     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
04078     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
04079     template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<165>, ZPZV<63>, ZPZV<215>,
          ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2»; };  // NOLINT
04080     template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type =
          POLYV<ZPZV<1>, ZPZV<265»; };  // NOLINT
04081     template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
          POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; };  // NOLINT
04082     template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<265»; };  // NOLINT
04083     template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6»; };  // NOLINT
04084     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
04085     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
04086     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
04087     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
04088     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
04089     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
04090     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
04091     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
04092     template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
          POLYV<ZPZV<1>, ZPZV<272»; };  // NOLINT
04093     template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
          POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; };  // NOLINT
04094     template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
```

```
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; };  // NOLINT
04095    template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<5»; };  // NOLINT
04096    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
04097    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
04098    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
04099    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
04100    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
04101    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
04102    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
04103    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
04104    template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
         POLYV<ZPZV<1>, ZPZV<278»; };  // NOLINT
04105    template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
         POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; };  // NOLINT
04106    template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<278»; };  // NOLINT
04107    template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3»; };  // NOLINT
04108    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
04109    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
04110    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
04111    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
04112    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
04113    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
04114    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
04115    template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<202>, ZPZV<68>, ZPZV<103>, ZPZV<116>,
         ZPZV<58>, ZPZV<28>, ZPZV<191>, ZPZV<3»; };  // NOLINT
04116    template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
         POLYV<ZPZV<1>, ZPZV<280»; };  // NOLINT
04117    template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
         POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3»; };  // NOLINT
04118    template<> struct ConwayPolynomial<283, 3> { using ZPZ = aerobus::zpz<283>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<280»; };  // NOLINT
04119    template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; };  // NOLINT
04120    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
04121    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
04122    template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<280»; };  // NOLINT
04123    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
04124    template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<65>, ZPZV<280»;
         };  // NOLINT
04125    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
04126    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
04127    template<> struct ConwayPolynomial<283, 12> { using ZPZ = aerobus::zpz<283>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<8>, ZPZV<96>, ZPZV<229>, ZPZV<49>,
         ZPZV<14>, ZPZV<56>, ZPZV<3»; };  // NOLINT
04128    template<> struct ConwayPolynomial<293, 1> { using ZPZ = aerobus::zpz<293>; using type =
         POLYV<ZPZV<1>, ZPZV<291»; };  // NOLINT
04129    template<> struct ConwayPolynomial<293, 2> { using ZPZ = aerobus::zpz<293>; using type =
         POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; };  // NOLINT
04130    template<> struct ConwayPolynomial<293, 3> { using ZPZ = aerobus::zpz<293>; using type =
```

```
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291»; };  // NOLINT
04131     template<> struct ConwayPolynomial<293, 4> { using ZPZ = aerobus::zpz<293>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2»; };  // NOLINT
04132     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
04133     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
04134     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
04135     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
04136     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
04137     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
04138     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
04139     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
04140     template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
        POLYV<ZPZV<1>, ZPZV<302»; };  // NOLINT
04141     template<> struct ConwayPolynomial<307, 2> { using ZPZ = aerobus::zpz<307>; using type =
        POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5»; };  // NOLINT
04142     template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<302»; };  // NOLINT
04143     template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5»; };  // NOLINT
04144     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
04145     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
04146     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
04147     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
04148     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
04149     template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
        POLYV<ZPZV<1>, ZPZV<294»; };  // NOLINT
04150     template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
        POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; };  // NOLINT
04151     template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; };  // NOLINT
04152     template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; };  // NOLINT
04153     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
04154     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
04155     template<> struct ConwayPolynomial<311, 7> { using ZPZ = aerobus::zpz<311>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<294»; };  // NOLINT
04156     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
04157     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
04158     template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<ZPZV<1>, ZPZV<303»; };  // NOLINT
04159     template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; };  // NOLINT
04160     template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; };  // NOLINT
04161     template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<239>, ZPZV<10»; };  // NOLINT
04162     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
04163     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
04164     template<> struct ConwayPolynomial<313, 7> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<303»; };  // NOLINT
04165     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
04166     template<> struct ConwayPolynomial<313, 9> { using ZPZ = aerobus::zpz<313>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<267>, ZPZV<300>, ZPZV<303»;
        };  // NOLINT
04167     template<> struct ConwayPolynomial<317, 1> { using ZPZ = aerobus::zpz<317>; using type =
        POLYV<ZPZV<1>, ZPZV<315»; };  // NOLINT
04168     template<> struct ConwayPolynomial<317, 2> { using ZPZ = aerobus::zpz<317>; using type =
```

```
          POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; };  // NOLINT
04169     template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<315»; };  // NOLINT
04170     template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2»; };  // NOLINT
04171     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
04172     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
04173     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
04174     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
04175     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
04176     template<> struct ConwayPolynomial<331, 1> { using ZPZ = aerobus::zpz<331>; using type =
          POLYV<ZPZV<1>, ZPZV<328»; };  // NOLINT
04177     template<> struct ConwayPolynomial<331, 2> { using ZPZ = aerobus::zpz<331>; using type =
          POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; };  // NOLINT
04178     template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<328»; };  // NOLINT
04179     template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<290>, ZPZV<3»; };  // NOLINT
04180     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
04181     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
04182     template<> struct ConwayPolynomial<331, 7> { using ZPZ = aerobus::zpz<331>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<328»; };  // NOLINT
04183     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
04184     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
04185     template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
          POLYV<ZPZV<1>, ZPZV<327»; };  // NOLINT
04186     template<> struct ConwayPolynomial<337, 2> { using ZPZ = aerobus::zpz<337>; using type =
          POLYV<ZPZV<1>, ZPZV<332>, ZPZV<10»; };  // NOLINT
04187     template<> struct ConwayPolynomial<337, 3> { using ZPZ = aerobus::zpz<337>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; };  // NOLINT
04188     template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10»; };  // NOLINT
04189     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
04190     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
04191     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
04192     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
04193     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
04194     template<> struct ConwayPolynomial<347, 1> { using ZPZ = aerobus::zpz<347>; using type =
          POLYV<ZPZV<1>, ZPZV<345»; };  // NOLINT
04195     template<> struct ConwayPolynomial<347, 2> { using ZPZ = aerobus::zpz<347>; using type =
          POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; };  // NOLINT
04196     template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<345»; };  // NOLINT
04197     template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; };  // NOLINT
04198     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
04199     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
04200     template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<345»; };  // NOLINT
04201     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
04202     template<> struct ConwayPolynomial<347, 9> { using ZPZ = aerobus::zpz<347>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<235>, ZPZV<252>, ZPZV<345»;
          };  // NOLINT
04203     template<> struct ConwayPolynomial<349, 1> { using ZPZ = aerobus::zpz<349>; using type =
          POLYV<ZPZV<1>, ZPZV<347»; };  // NOLINT
04204     template<> struct ConwayPolynomial<349, 2> { using ZPZ = aerobus::zpz<349>; using type =
          POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; };  // NOLINT
04205     template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; };  // NOLINT
04206     template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2»; };  // NOLINT
04207     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
```

```
04208     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
04209     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
04210     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
04211     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
04212     template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<350>; };  // NOLINT
04213     template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3>; };  // NOLINT
04214     template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<350>; };  // NOLINT
04215     template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<3>; };  // NOLINT
04216     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
04217     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
04218     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
04219     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
04220     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
04221     template<> struct ConwayPolynomial<359, 1> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<352>; };  // NOLINT
04222     template<> struct ConwayPolynomial<359, 2> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7>; };  // NOLINT
04223     template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<352>; };  // NOLINT
04224     template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<229>, ZPZV<7>; };  // NOLINT
04225     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
04226     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
04227     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
04228     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
04229     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
04230     template<> struct ConwayPolynomial<367, 1> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<361>; };  // NOLINT
04231     template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6>; };  // NOLINT
04232     template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<361>; };  // NOLINT
04233     template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6>; };  // NOLINT
04234     template<> struct ConwayPolynomial<367, 5> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<361>; };  // NOLINT
04235     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
04236     template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<361>; };  // NOLINT
04237     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
04238     template<> struct ConwayPolynomial<367, 9> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<213>, ZPZV<268>, ZPZV<361>;
      };  // NOLINT
04239     template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<371>; };  // NOLINT
04240     template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2>; };  // NOLINT
04241     template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<371>; };  // NOLINT
04242     template<> struct ConwayPolynomial<373, 4> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304>, ZPZV<2>; };  // NOLINT
04243     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
04244     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
04245     template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<371>; };  // NOLINT
04246     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
```

```
04247      template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<238>, ZPZV<370>, ZPZV<371>;
      }; // NOLINT
04248      template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<377>; }; // NOLINT
04249      template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2>; }; // NOLINT
04250      template<> struct ConwayPolynomial<379, 3> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377>; }; // NOLINT
04251      template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327>, ZPZV<2>; }; // NOLINT
04252      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
04253      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
04254      template<> struct ConwayPolynomial<379, 7> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<377>; }; // NOLINT
04255      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
04256      template<> struct ConwayPolynomial<379, 9> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<362>, ZPZV<369>, ZPZV<377>;
      }; // NOLINT
04257      template<> struct ConwayPolynomial<383, 1> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<378>; }; // NOLINT
04258      template<> struct ConwayPolynomial<383, 2> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5>; }; // NOLINT
04259      template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<378>; }; // NOLINT
04260      template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5>; }; // NOLINT
04261      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
04262      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
04263      template<> struct ConwayPolynomial<383, 7> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<378>; }; // NOLINT
04264      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
04265      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
04266      template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<387>; }; // NOLINT
04267      template<> struct ConwayPolynomial<389, 2> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<379>, ZPZV<2>; }; // NOLINT
04268      template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387>; }; // NOLINT
04269      template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<266>, ZPZV<2>; }; // NOLINT
04270      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
04271      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
04272      template<> struct ConwayPolynomial<389, 7> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<387>; }; // NOLINT
04273      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
04274      template<> struct ConwayPolynomial<389, 9> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<308>, ZPZV<387>;
      }; // NOLINT
04275      template<> struct ConwayPolynomial<397, 1> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<392>; }; // NOLINT
04276      template<> struct ConwayPolynomial<397, 2> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5>; }; // NOLINT
04277      template<> struct ConwayPolynomial<397, 3> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<392>; }; // NOLINT
04278      template<> struct ConwayPolynomial<397, 4> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5>; }; // NOLINT
04279      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
04280      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
04281      template<> struct ConwayPolynomial<397, 7> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<392>; }; // NOLINT
04282      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
04283      template<> struct ConwayPolynomial<397, 9> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<166>, ZPZV<252>, ZPZV<392>;
      }; // NOLINT
04284      template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<398>; }; // NOLINT
04285      template<> struct ConwayPolynomial<401, 2> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3>; }; // NOLINT
```

```
04286     template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<398»; }; // NOLINT
04287     template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<3»; }; // NOLINT
04288     template<> struct ConwayPolynomial<401, 5> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<398»; }; // NOLINT
04289     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
04290     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
04291     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
04292     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
04293     template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
04294     template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT
04295     template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388»; }; // NOLINT
04296     template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<407>, ZPZV<21»; }; // NOLINT
04297     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
04298     template<> struct ConwayPolynomial<409, 6> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<372>, ZPZV<53>, ZPZV<364>, ZPZV<21»; }; // NOLINT
04299     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
04300     template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<256>, ZPZV<69>, ZPZV<396>, ZPZV<21»; }; //
      NOLINT
04301     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
04302     template<> struct ConwayPolynomial<419, 1> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<417»; }; // NOLINT
04303     template<> struct ConwayPolynomial<419, 2> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT
04304     template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<417»; }; // NOLINT
04305     template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<373>, ZPZV<2»; }; // NOLINT
04306     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
04307     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
04308     template<> struct ConwayPolynomial<419, 7> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; // NOLINT
04309     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
04310     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
04311     template<> struct ConwayPolynomial<421, 1> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<419»; }; // NOLINT
04312     template<> struct ConwayPolynomial<421, 2> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT
04313     template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419»; }; // NOLINT
04314     template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2»; }; // NOLINT
04315     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
04316     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
04317     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
04318     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
04319     template<> struct ConwayPolynomial<421, 9> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<394>, ZPZV<145>, ZPZV<419»;
      }; // NOLINT
04320     template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
04321     template<> struct ConwayPolynomial<431, 2> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7»; }; // NOLINT
04322     template<> struct ConwayPolynomial<431, 3> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424»; }; // NOLINT
04323     template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7»; }; // NOLINT
04324     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
04325     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
04326      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
04327      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
04328      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
04329      template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
           POLYV<ZPZV<1>, ZPZV<428»; };  // NOLINT
04330      template<> struct ConwayPolynomial<433, 2> { using ZPZ = aerobus::zpz<433>; using type =
           POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5»; };  // NOLINT
04331      template<> struct ConwayPolynomial<433, 3> { using ZPZ = aerobus::zpz<433>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428»; };  // NOLINT
04332      template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<402>, ZPZV<5»; };  // NOLINT
04333      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
04334      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
04335      template<> struct ConwayPolynomial<433, 7> { using ZPZ = aerobus::zpz<433>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<428»; };  // NOLINT
04336      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
04337      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
04338      template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
           POLYV<ZPZV<1>, ZPZV<424»; };  // NOLINT
04339      template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
           POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; };  // NOLINT
04340      template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424»; };  // NOLINT
04341      template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; };  // NOLINT
04342      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
04343      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
04344      template<> struct ConwayPolynomial<439, 7> { using ZPZ = aerobus::zpz<439>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; };  // NOLINT
04345      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
04346      template<> struct ConwayPolynomial<439, 9> { using ZPZ = aerobus::zpz<439>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<342>, ZPZV<254>, ZPZV<424»;
           };  // NOLINT
04347      template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
           POLYV<ZPZV<1>, ZPZV<441»; };  // NOLINT
04348      template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
           POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; };  // NOLINT
04349      template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<441»; };  // NOLINT
04350      template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2»; };  // NOLINT
04351      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
04352      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
04353      template<> struct ConwayPolynomial<443, 7> { using ZPZ = aerobus::zpz<443>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<441»; };  // NOLINT
04354      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
04355      template<> struct ConwayPolynomial<443, 9> { using ZPZ = aerobus::zpz<443>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<125>, ZPZV<109>, ZPZV<441»;
           };  // NOLINT
04356      template<> struct ConwayPolynomial<449, 1> { using ZPZ = aerobus::zpz<449>; using type =
           POLYV<ZPZV<1>, ZPZV<446»; };  // NOLINT
04357      template<> struct ConwayPolynomial<449, 2> { using ZPZ = aerobus::zpz<449>; using type =
           POLYV<ZPZV<1>, ZPZV<444>, ZPZV<3»; };  // NOLINT
04358      template<> struct ConwayPolynomial<449, 3> { using ZPZ = aerobus::zpz<449>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446»; };  // NOLINT
04359      template<> struct ConwayPolynomial<449, 4> { using ZPZ = aerobus::zpz<449>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3»; };  // NOLINT
04360      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
04361      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
04362      template<> struct ConwayPolynomial<449, 7> { using ZPZ = aerobus::zpz<449>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<446»; };  // NOLINT
04363      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
04364      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
04365     template<> struct ConwayPolynomial<457, 1> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<444»; };  // NOLINT
04366     template<> struct ConwayPolynomial<457, 2> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<454>, ZPZV<13»; };  // NOLINT
04367     template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444»; };  // NOLINT
04368     template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13»; };  // NOLINT
04369     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
04370     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
04371     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
04372     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
04373     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
04374     template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<459»; };  // NOLINT
04375     template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2»; };  // NOLINT
04376     template<> struct ConwayPolynomial<461, 3> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459»; };  // NOLINT
04377     template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2»; };  // NOLINT
04378     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
04379     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
04380     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
04381     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
04382     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
04383     template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<460»; };  // NOLINT
04384     template<> struct ConwayPolynomial<463, 2> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; };  // NOLINT
04385     template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<460»; };  // NOLINT
04386     template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; };  // NOLINT
04387     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
04388     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
04389     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
04390     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
04391     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
04392     template<> struct ConwayPolynomial<467, 1> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<465»; };  // NOLINT
04393     template<> struct ConwayPolynomial<467, 2> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; };  // NOLINT
04394     template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465»; };  // NOLINT
04395     template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; };  // NOLINT
04396     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
04397     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
04398     template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<465»; };  // NOLINT
04399     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
04400     template<> struct ConwayPolynomial<467, 9> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<397>, ZPZV<447>, ZPZV<465»;
      };  // NOLINT
04401     template<> struct ConwayPolynomial<479, 1> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<466»; };  // NOLINT
04402     template<> struct ConwayPolynomial<479, 2> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; };  // NOLINT
04403     template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
```

```
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466»; };  // NOLINT
04404     template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13»; };  // NOLINT
04405     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
04406     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
04407     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
04408     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
04409     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
04410     template<> struct ConwayPolynomial<487, 1> { using ZPZ = aerobus::zpz<487>; using type =
          POLYV<ZPZV<1>, ZPZV<484»; };  // NOLINT
04411     template<> struct ConwayPolynomial<487, 2> { using ZPZ = aerobus::zpz<487>; using type =
          POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; };  // NOLINT
04412     template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; };  // NOLINT
04413     template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3»; };  // NOLINT
04414     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
04415     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
04416     template<> struct ConwayPolynomial<487, 7> { using ZPZ = aerobus::zpz<487>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<484»; };  // NOLINT
04417     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
04418     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
04419     template<> struct ConwayPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
          POLYV<ZPZV<1>, ZPZV<489»; };  // NOLINT
04420     template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
          POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; };  // NOLINT
04421     template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<489»; };  // NOLINT
04422     template<> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360>, ZPZV<2»; };  // NOLINT
04423     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
04424     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
04425     template<> struct ConwayPolynomial<491, 7> { using ZPZ = aerobus::zpz<491>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; };  // NOLINT
04426     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
04427     template<> struct ConwayPolynomial<491, 9> { using ZPZ = aerobus::zpz<491>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<453>, ZPZV<489»;
          };  // NOLINT
04428     template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
          POLYV<ZPZV<1>, ZPZV<492»; };  // NOLINT
04429     template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
          POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; };  // NOLINT
04430     template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; };  // NOLINT
04431     template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; };  // NOLINT
04432     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
04433     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
04434     template<> struct ConwayPolynomial<499, 7> { using ZPZ = aerobus::zpz<499>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<492»; };  // NOLINT
04435     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
04436     template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<491>, ZPZV<222>, ZPZV<492»;
          };  // NOLINT
04437     template<> struct ConwayPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
          POLYV<ZPZV<1>, ZPZV<498»; };  // NOLINT
04438     template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
          POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; };  // NOLINT
04439     template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<498»; };  // NOLINT
04440     template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<325>, ZPZV<5»; };  // NOLINT
04441     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
04442     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
```

```
04443     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
04444     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
04445     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
04446     template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
     POLYV<ZPZV<1>, ZPZV<507»; };  // NOLINT
04447     template<> struct ConwayPolynomial<509, 2> { using ZPZ = aerobus::zpz<509>; using type =
     POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2»; };  // NOLINT
04448     template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<507»; };  // NOLINT
04449     template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2»; };  // NOLINT
04450     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
04451     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
04452     template<> struct ConwayPolynomial<509, 7> { using ZPZ = aerobus::zpz<509>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<507»; };  // NOLINT
04453     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
04454     template<> struct ConwayPolynomial<509, 9> { using ZPZ = aerobus::zpz<509>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<314>, ZPZV<28>, ZPZV<507»;
     };  // NOLINT
04455     template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
     POLYV<ZPZV<1>, ZPZV<518»; };  // NOLINT
04456     template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
     POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; };  // NOLINT
04457     template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518»; };  // NOLINT
04458     template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<509>, ZPZV<3»; };  // NOLINT
04459     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
04460     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
04461     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
04462     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
04463     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
04464     template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
     POLYV<ZPZV<1>, ZPZV<521»; };  // NOLINT
04465     template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
     POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; };  // NOLINT
04466     template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<521»; };  // NOLINT
04467     template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2»; };  // NOLINT
04468     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
04469     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
04470     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
04471     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
04472     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
04473     template<> struct ConwayPolynomial<541, 1> { using ZPZ = aerobus::zpz<541>; using type =
     POLYV<ZPZV<1>, ZPZV<539»; };  // NOLINT
04474     template<> struct ConwayPolynomial<541, 2> { using ZPZ = aerobus::zpz<541>; using type =
     POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; };  // NOLINT
04475     template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539»; };  // NOLINT
04476     template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2»; };  // NOLINT
04477     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
04478     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
04479     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
04480     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
04481     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
04482     template<> struct ConwayPolynomial<547, 1> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<545»; };  // NOLINT
04483     template<> struct ConwayPolynomial<547, 2> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; };  // NOLINT
04484     template<> struct ConwayPolynomial<547, 3> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<545»; };  // NOLINT
04485     template<> struct ConwayPolynomial<547, 4> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<334>, ZPZV<2»; };  // NOLINT
04486     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
04487     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
04488     template<> struct ConwayPolynomial<547, 7> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<545»; };  // NOLINT
04489     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
04490     template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<238>, ZPZV<263>, ZPZV<545»;
      };  // NOLINT
04491     template<> struct ConwayPolynomial<557, 1> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<555»; };  // NOLINT
04492     template<> struct ConwayPolynomial<557, 2> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; };  // NOLINT
04493     template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<555»; };  // NOLINT
04494     template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; };  // NOLINT
04495     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
04496     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
04497     template<> struct ConwayPolynomial<557, 7> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<555»; };  // NOLINT
04498     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
04499     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
04500     template<> struct ConwayPolynomial<563, 1> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<561»; };  // NOLINT
04501     template<> struct ConwayPolynomial<563, 2> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; };  // NOLINT
04502     template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; };  // NOLINT
04503     template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<20>, ZPZV<399>, ZPZV<2»; };  // NOLINT
04504     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
04505     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
04506     template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<561»; };  // NOLINT
04507     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
04508     template<> struct ConwayPolynomial<563, 9> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<15>, ZPZV<19>, ZPZV<561»; };
      // NOLINT
04509     template<> struct ConwayPolynomial<569, 1> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<566»; };  // NOLINT
04510     template<> struct ConwayPolynomial<569, 2> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; };  // NOLINT
04511     template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; };  // NOLINT
04512     template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; };  // NOLINT
04513     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
04514     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
04515     template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<566»; };  // NOLINT
04516     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
04517     template<> struct ConwayPolynomial<569, 9> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<478>, ZPZV<566>, ZPZV<566»;
      };  // NOLINT
04518     template<> struct ConwayPolynomial<571, 1> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<568»; };  // NOLINT
04519     template<> struct ConwayPolynomial<571, 2> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; };  // NOLINT
04520     template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<568»; };  // NOLINT
```

```
04521    template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3»; };  // NOLINT
04522    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
04523    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
04524    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
04525    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
04526    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
04527    template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
     POLYV<ZPZV<1>, ZPZV<572»; };  // NOLINT
04528    template<> struct ConwayPolynomial<577, 2> { using ZPZ = aerobus::zpz<577>; using type =
     POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5»; };  // NOLINT
04529    template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572»; };  // NOLINT
04530    template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<5»; };  // NOLINT
04531    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
04532    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
04533    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
04534    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
04535    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
04536    template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
     POLYV<ZPZV<1>, ZPZV<585»; };  // NOLINT
04537    template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
     POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2»; };  // NOLINT
04538    template<> struct ConwayPolynomial<587, 3> { using ZPZ = aerobus::zpz<587>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585»; };  // NOLINT
04539    template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<444>, ZPZV<2»; };  // NOLINT
04540    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
04541    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
04542    template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<585»; };  // NOLINT
04543    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
04544    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
04545    template<> struct ConwayPolynomial<593, 1> { using ZPZ = aerobus::zpz<593>; using type =
     POLYV<ZPZV<1>, ZPZV<590»; };  // NOLINT
04546    template<> struct ConwayPolynomial<593, 2> { using ZPZ = aerobus::zpz<593>; using type =
     POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3»; };  // NOLINT
04547    template<> struct ConwayPolynomial<593, 3> { using ZPZ = aerobus::zpz<593>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<590»; };  // NOLINT
04548    template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3»; };  // NOLINT
04549    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
04550    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
04551    template<> struct ConwayPolynomial<593, 7> { using ZPZ = aerobus::zpz<593>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<590»; };  // NOLINT
04552    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
04553    template<> struct ConwayPolynomial<593, 9> { using ZPZ = aerobus::zpz<593>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223>, ZPZV<523>, ZPZV<590»;
     };  // NOLINT
04554    template<> struct ConwayPolynomial<599, 1> { using ZPZ = aerobus::zpz<599>; using type =
     POLYV<ZPZV<1>, ZPZV<592»; };  // NOLINT
04555    template<> struct ConwayPolynomial<599, 2> { using ZPZ = aerobus::zpz<599>; using type =
     POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; };  // NOLINT
04556    template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<592»; };  // NOLINT
04557    template<> struct ConwayPolynomial<599, 4> { using ZPZ = aerobus::zpz<599>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7»; };  // NOLINT
04558    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
04559    template<> struct ConwayPolynomial<599, 6> { using ZPZ = aerobus::zpz<599>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<515>, ZPZV<274>, ZPZV<586>, ZPZV<7»; };  // NOLINT
04560    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
04561    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
04562    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
04563    template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type =
         POLYV<ZPZV<1>, ZPZV<594»; };  // NOLINT
04564    template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
         POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; };  // NOLINT
04565    template<> struct ConwayPolynomial<601, 3> { using ZPZ = aerobus::zpz<601>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<594»; };  // NOLINT
04566    template<> struct ConwayPolynomial<601, 4> { using ZPZ = aerobus::zpz<601>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; };  // NOLINT
04567    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
04568    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
04569    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
04570    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
04571    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
04572    template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<604»; };  // NOLINT
04573    template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; };  // NOLINT
04574    template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<604»; };  // NOLINT
04575    template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3»; };  // NOLINT
04576    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
04577    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
04578    template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<604»; };  // NOLINT
04579    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
04580    template<> struct ConwayPolynomial<607, 9> { using ZPZ = aerobus::zpz<607>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<444>, ZPZV<129>, ZPZV<604»;
         };  // NOLINT
04581    template<> struct ConwayPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<611»; };  // NOLINT
04582    template<> struct ConwayPolynomial<613, 2> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; };  // NOLINT
04583    template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<611»; };  // NOLINT
04584    template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2»; };  // NOLINT
04585    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
04586    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
04587    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
04588    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
04589    template<> struct ConwayPolynomial<613, 9> { using ZPZ = aerobus::zpz<613>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<513>, ZPZV<536>, ZPZV<611»;
         };  // NOLINT
04590    template<> struct ConwayPolynomial<617, 1> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<614»; };  // NOLINT
04591    template<> struct ConwayPolynomial<617, 2> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; };  // NOLINT
04592    template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<614»; };  // NOLINT
04593    template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3»; };  // NOLINT
04594    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
04595    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
04596    template<> struct ConwayPolynomial<617, 7> { using ZPZ = aerobus::zpz<617>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<614»; };  // NOLINT
04597    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
04598    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
```

```
04599     template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<617»; };  // NOLINT
04600     template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; };  // NOLINT
04601     template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617»; };  // NOLINT
04602     template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492>, ZPZV<2»; };  // NOLINT
04603     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
04604     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
04605     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
04606     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
04607     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
04608     template<> struct ConwayPolynomial<631, 1> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<628»; };  // NOLINT
04609     template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3»; };  // NOLINT
04610     template<> struct ConwayPolynomial<631, 3> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<628»; };  // NOLINT
04611     template<> struct ConwayPolynomial<631, 4> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<3»; };  // NOLINT
04612     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
04613     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
04614     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
04615     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
04616     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
04617     template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<638»; };  // NOLINT
04618     template<> struct ConwayPolynomial<641, 2> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3»; };  // NOLINT
04619     template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<638»; };  // NOLINT
04620     template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<3»; };  // NOLINT
04621     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
04622     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
04623     template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<638»; };  // NOLINT
04624     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
04625     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
04626     template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<632»; };  // NOLINT
04627     template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11»; };  // NOLINT
04628     template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<632»; };  // NOLINT
04629     template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<600>, ZPZV<11»; };  // NOLINT
04630     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
04631     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
04632     template<> struct ConwayPolynomial<643, 7> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<632»; };  // NOLINT
04633     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
04634     template<> struct ConwayPolynomial<643, 9> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<591>, ZPZV<475>, ZPZV<632»;
      };  // NOLINT
04635     template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<642»; };  // NOLINT
04636     template<> struct ConwayPolynomial<647, 2> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5»; };  // NOLINT
04637     template<> struct ConwayPolynomial<647, 3> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<642»; };  // NOLINT
04638     template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
```

```
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5>; };  // NOLINT
04639     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
04640     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
04641     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
04642     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
04643     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
04644     template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
       POLYV<ZPZV<1>, ZPZV<651>; };  // NOLINT
04645     template<> struct ConwayPolynomial<653, 2> { using ZPZ = aerobus::zpz<653>; using type =
       POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2>; };  // NOLINT
04646     template<> struct ConwayPolynomial<653, 3> { using ZPZ = aerobus::zpz<653>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<651>; };  // NOLINT
04647     template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2>; };  // NOLINT
04648     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
04649     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
04650     template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<651>; };  // NOLINT
04651     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
04652     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
04653     template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
       POLYV<ZPZV<1>, ZPZV<657>; };  // NOLINT
04654     template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
       POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2>; };  // NOLINT
04655     template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<657>; };  // NOLINT
04656     template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2>; };  // NOLINT
04657     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
04658     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
04659     template<> struct ConwayPolynomial<659, 7> { using ZPZ = aerobus::zpz<659>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<657>; };  // NOLINT
04660     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
04661     template<> struct ConwayPolynomial<659, 9> { using ZPZ = aerobus::zpz<659>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592>, ZPZV<46>, ZPZV<657>;
       };  // NOLINT
04662     template<> struct ConwayPolynomial<661, 1> { using ZPZ = aerobus::zpz<661>; using type =
       POLYV<ZPZV<1>, ZPZV<659>; };  // NOLINT
04663     template<> struct ConwayPolynomial<661, 2> { using ZPZ = aerobus::zpz<661>; using type =
       POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2>; };  // NOLINT
04664     template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<659>; };  // NOLINT
04665     template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2>; };  // NOLINT
04666     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
04667     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
04668     template<> struct ConwayPolynomial<661, 7> { using ZPZ = aerobus::zpz<661>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<659>; };  // NOLINT
04669     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
04670     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
04671     template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<668>; };  // NOLINT
04672     template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5>; };  // NOLINT
04673     template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<668>; };  // NOLINT
04674     template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5>; };  // NOLINT
04675     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
04676     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
04677     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
```

```
04678    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
04679    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
04680    template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
     POLYV<ZPZV<1>, ZPZV<675»; };  // NOLINT
04681    template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
     POLYV<ZPZV<1>, ZPZV<672>, ZPZV<2»; };  // NOLINT
04682    template<> struct ConwayPolynomial<677, 3> { using ZPZ = aerobus::zpz<677>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<675»; };  // NOLINT
04683    template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2»; };  // NOLINT
04684    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
04685    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
04686    template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<675»; };  // NOLINT
04687    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
04688    template<> struct ConwayPolynomial<677, 9> { using ZPZ = aerobus::zpz<677>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<504>, ZPZV<404>, ZPZV<675»;
     };  // NOLINT
04689    template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
     POLYV<ZPZV<1>, ZPZV<678»; };  // NOLINT
04690    template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
     POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; };  // NOLINT
04691    template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<678»; };  // NOLINT
04692    template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5»; };  // NOLINT
04693    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
04694    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
04695    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
04696    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
04697    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
04698    template<> struct ConwayPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
     POLYV<ZPZV<1>, ZPZV<688»; };  // NOLINT
04699    template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
     POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; };  // NOLINT
04700    template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<688»; };  // NOLINT
04701    template<> struct ConwayPolynomial<691, 4> { using ZPZ = aerobus::zpz<691>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; };  // NOLINT
04702    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
04703    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
04704    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
04705    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
04706    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
04707    template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
     POLYV<ZPZV<1>, ZPZV<699»; };  // NOLINT
04708    template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
     POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; };  // NOLINT
04709    template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; };  // NOLINT
04710    template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2»; };  // NOLINT
04711    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
04712    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
04713    template<> struct ConwayPolynomial<701, 7> { using ZPZ = aerobus::zpz<701>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<699»; };  // NOLINT
04714    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
04715    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
04716    template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
```

```
           POLYV<ZPZV<1>, ZPZV<707»; };  // NOLINT
04717       template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
           POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; };  // NOLINT
04718       template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707»; };  // NOLINT
04719       template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; };  // NOLINT
04720       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
04721       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
04722       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
04723       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
04724       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
04725       template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
           POLYV<ZPZV<1>, ZPZV<708»; };  // NOLINT
04726       template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
           POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; };  // NOLINT
04727       template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708»; };  // NOLINT
04728       template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<602>, ZPZV<11»; };  // NOLINT
04729       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
04730       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
04731       template<> struct ConwayPolynomial<719, 7> { using ZPZ = aerobus::zpz<719>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<708»; };  // NOLINT
04732       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
04733       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
04734       template<> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
           POLYV<ZPZV<1>, ZPZV<722»; };  // NOLINT
04735       template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
           POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; };  // NOLINT
04736       template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<722»; };  // NOLINT
04737       template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5»; };  // NOLINT
04738       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
04739       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
04740       template<> struct ConwayPolynomial<727, 7> { using ZPZ = aerobus::zpz<727>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<722»; };  // NOLINT
04741       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
04742       template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<573>, ZPZV<502>, ZPZV<722»;
           };  // NOLINT
04743       template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
           POLYV<ZPZV<1>, ZPZV<727»; };  // NOLINT
04744       template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
           POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; };  // NOLINT
04745       template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727»; };  // NOLINT
04746       template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<6»; };  // NOLINT
04747       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
04748       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
04749       template<> struct ConwayPolynomial<733, 7> { using ZPZ = aerobus::zpz<733>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<727»; };  // NOLINT
04750       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
04751       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
04752       template<> struct ConwayPolynomial<739, 1> { using ZPZ = aerobus::zpz<739>; using type =
           POLYV<ZPZV<1>, ZPZV<736»; };  // NOLINT
04753       template<> struct ConwayPolynomial<739, 2> { using ZPZ = aerobus::zpz<739>; using type =
           POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; };  // NOLINT
04754       template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; };  // NOLINT
04755       template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<3»; };  // NOLINT
```

```
04756     template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<736»; }; // NOLINT
04757     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
04758     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
04759     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
04760     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
04761     template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
     POLYV<ZPZV<1>, ZPZV<738»; }; // NOLINT
04762     template<> struct ConwayPolynomial<743, 2> { using ZPZ = aerobus::zpz<743>; using type =
     POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; }; // NOLINT
04763     template<> struct ConwayPolynomial<743, 3> { using ZPZ = aerobus::zpz<743>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<738»; }; // NOLINT
04764     template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<5»; }; // NOLINT
04765     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
04766     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
04767     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
04768     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
04769     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
04770     template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
     POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
04771     template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
     POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT
04772     template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; }; // NOLINT
04773     template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<525>, ZPZV<3»; }; // NOLINT
04774     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
04775     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
04776     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
04777     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
04778     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
04779     template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
     POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
04780     template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
     POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2»; }; // NOLINT
04781     template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT
04782     template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; }; // NOLINT
04783     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
04784     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
04785     template<> struct ConwayPolynomial<757, 7> { using ZPZ = aerobus::zpz<757>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<755»; }; // NOLINT
04786     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
04787     template<> struct ConwayPolynomial<757, 9> { using ZPZ = aerobus::zpz<757>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<688>, ZPZV<702>, ZPZV<755»;
     }; // NOLINT
04788     template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
     POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
04789     template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
     POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; }; // NOLINT
04790     template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755»; }; // NOLINT
04791     template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<6»; }; // NOLINT
04792     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
04793     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
04794     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
04795     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
04796    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
04797    template<> struct ConwayPolynomial<769, 1> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<758»; }; // NOLINT
04798    template<> struct ConwayPolynomial<769, 2> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; }; // NOLINT
04799    template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; }; // NOLINT
04800    template<> struct ConwayPolynomial<769, 4> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; }; // NOLINT
04801    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
04802    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
04803    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
04804    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
04805    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
04806    template<> struct ConwayPolynomial<773, 1> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<771»; }; // NOLINT
04807    template<> struct ConwayPolynomial<773, 2> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»; }; // NOLINT
04808    template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<771»; }; // NOLINT
04809    template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2»; }; // NOLINT
04810    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
04811    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
04812    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
04813    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
04814    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
04815    template<> struct ConwayPolynomial<787, 1> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<785»; }; // NOLINT
04816    template<> struct ConwayPolynomial<787, 2> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
04817    template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785»; }; // NOLINT
04818    template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2»; }; // NOLINT
04819    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
04820    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
04821    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
04822    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
04823    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
04824    template<> struct ConwayPolynomial<797, 1> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<795»; }; // NOLINT
04825    template<> struct ConwayPolynomial<797, 2> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»; }; // NOLINT
04826    template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<795»; }; // NOLINT
04827    template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<717>, ZPZV<2»; }; // NOLINT
04828    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
04829    template<> struct ConwayPolynomial<797, 6> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<657>, ZPZV<396>, ZPZV<71>, ZPZV<2»; }; // NOLINT
04830    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
04831    template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2»; }; //
      NOLINT
04832    template<> struct ConwayPolynomial<797, 9> { using ZPZ = aerobus::zpz<797>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<240>, ZPZV<599>, ZPZV<795»;
      }; // NOLINT
04833    template<> struct ConwayPolynomial<809, 1> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<806»; }; // NOLINT
```

```
04834     template<> struct ConwayPolynomial<809, 2> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
04835     template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<806»; }; // NOLINT
04836     template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; }; // NOLINT
04837     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
04838     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
04839     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
04840     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
04841     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
04842     template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
04843     template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<806>, ZPZV<3»; }; // NOLINT
04844     template<> struct ConwayPolynomial<811, 3> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<808»; }; // NOLINT
04845     template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3»; }; // NOLINT
04846     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
04847     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
04848     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
04849     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
04850     template<> struct ConwayPolynomial<811, 9> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<382>, ZPZV<200>, ZPZV<808»;
      }; // NOLINT
04851     template<> struct ConwayPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<819»; }; // NOLINT
04852     template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT
04853     template<> struct ConwayPolynomial<821, 3> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<819»; }; // NOLINT
04854     template<> struct ConwayPolynomial<821, 4> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<662>, ZPZV<2»; }; // NOLINT
04855     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
04856     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
04857     template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<819»; }; // NOLINT
04858     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
04859     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
04860     template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
04861     template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; }; // NOLINT
04862     template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
04863     template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT
04864     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
04865     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
04866     template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<820»; }; // NOLINT
04867     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
04868     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
04869     template<> struct ConwayPolynomial<827, 1> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<825»; }; // NOLINT
04870     template<> struct ConwayPolynomial<827, 2> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT
04871     template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825»; }; // NOLINT
04872     template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2»; }; // NOLINT
04873     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
04874     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
04875     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
04876     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
04877     template<> struct ConwayPolynomial<827, 9> { using ZPZ = aerobus::zpz<827>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<372>, ZPZV<825»;
          };  // NOLINT
04878     template<> struct ConwayPolynomial<829, 1> { using ZPZ = aerobus::zpz<829>; using type =
          POLYV<ZPZV<1>, ZPZV<827»; };  // NOLINT
04879     template<> struct ConwayPolynomial<829, 2> { using ZPZ = aerobus::zpz<829>; using type =
          POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; };  // NOLINT
04880     template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<827»; };  // NOLINT
04881     template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<604>, ZPZV<2»; };  // NOLINT
04882     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
04883     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
04884     template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<827»; };  // NOLINT
04885     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
04886     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
04887     template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
          POLYV<ZPZV<1>, ZPZV<828»; };  // NOLINT
04888     template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
          POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; };  // NOLINT
04889     template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; };  // NOLINT
04890     template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11»; };  // NOLINT
04891     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
04892     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
04893     template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<828»; };  // NOLINT
04894     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
04895     template<> struct ConwayPolynomial<839, 9> { using ZPZ = aerobus::zpz<839>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<349>, ZPZV<206>, ZPZV<828»;
          };  // NOLINT
04896     template<> struct ConwayPolynomial<853, 1> { using ZPZ = aerobus::zpz<853>; using type =
          POLYV<ZPZV<1>, ZPZV<851»; };  // NOLINT
04897     template<> struct ConwayPolynomial<853, 2> { using ZPZ = aerobus::zpz<853>; using type =
          POLYV<ZPZV<1>, ZPZV<852>, ZPZV<2»; };  // NOLINT
04898     template<> struct ConwayPolynomial<853, 3> { using ZPZ = aerobus::zpz<853>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851»; };  // NOLINT
04899     template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<2»; };  // NOLINT
04900     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
04901     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
04902     template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<851»; };  // NOLINT
04903     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
04904     template<> struct ConwayPolynomial<853, 9> { using ZPZ = aerobus::zpz<853>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<677>, ZPZV<821>, ZPZV<851»;
          };  // NOLINT
04905     template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
          POLYV<ZPZV<1>, ZPZV<854»; };  // NOLINT
04906     template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
          POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; };  // NOLINT
04907     template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; };  // NOLINT
04908     template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<528>, ZPZV<3»; };  // NOLINT
04909     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
04910     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
04911     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
04912     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
04913     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
04914     template<> struct ConwayPolynomial<859, 1> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<857»; }; // NOLINT
04915     template<> struct ConwayPolynomial<859, 2> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT
04916     template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857»; }; // NOLINT
04917     template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2»; }; // NOLINT
04918     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
04919     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
04920     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
04921     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
04922     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
04923     template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<858»; }; // NOLINT
04924     template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; }; // NOLINT
04925     template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<858»; }; // NOLINT
04926     template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5»; }; // NOLINT
04927     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
04928     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
04929     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
04930     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
04931     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
04932     template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
04933     template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; }; // NOLINT
04934     template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875»; }; // NOLINT
04935     template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; }; // NOLINT
04936     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
04937     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
04938     template<> struct ConwayPolynomial<877, 7> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<875»; }; // NOLINT
04939     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
04940     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
04941     template<> struct ConwayPolynomial<881, 1> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<878»; }; // NOLINT
04942     template<> struct ConwayPolynomial<881, 2> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT
04943     template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<878»; }; // NOLINT
04944     template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<447>, ZPZV<3»; }; // NOLINT
04945     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
04946     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
04947     template<> struct ConwayPolynomial<881, 7> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<878»; }; // NOLINT
04948     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
04949     template<> struct ConwayPolynomial<881, 9> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<587>, ZPZV<510>, ZPZV<878»;
      }; // NOLINT
04950     template<> struct ConwayPolynomial<883, 1> { using ZPZ = aerobus::zpz<883>; using type =
      POLYV<ZPZV<1>, ZPZV<881»; }; // NOLINT
04951     template<> struct ConwayPolynomial<883, 2> { using ZPZ = aerobus::zpz<883>; using type =
```

```
           POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; };   // NOLINT
04952      template<> struct ConwayPolynomial<883, 3> { using ZPZ = aerobus::zpz<883>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<881»; };   // NOLINT
04953      template<> struct ConwayPolynomial<883, 4> { using ZPZ = aerobus::zpz<883>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2»; };   // NOLINT
04954      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
04955      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
04956      template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<881»; };   // NOLINT
04957      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
04958      template<> struct ConwayPolynomial<883, 9> { using ZPZ = aerobus::zpz<883>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<360>, ZPZV<557>, ZPZV<881»;
           };   // NOLINT
04959      template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
           POLYV<ZPZV<1>, ZPZV<882»; };   // NOLINT
04960      template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
           POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; };   // NOLINT
04961      template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<882»; };   // NOLINT
04962      template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5»; };   // NOLINT
04963      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
04964      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
04965      template<> struct ConwayPolynomial<887, 7> { using ZPZ = aerobus::zpz<887>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<882»; };   // NOLINT
04966      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
04967      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
04968      template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
           POLYV<ZPZV<1>, ZPZV<905»; };   // NOLINT
04969      template<> struct ConwayPolynomial<907, 2> { using ZPZ = aerobus::zpz<907>; using type =
           POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2»; };   // NOLINT
04970      template<> struct ConwayPolynomial<907, 3> { using ZPZ = aerobus::zpz<907>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<905»; };   // NOLINT
04971      template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2»; };   // NOLINT
04972      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
04973      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
04974      template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; };   // NOLINT
04975      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
04976      template<> struct ConwayPolynomial<907, 9> { using ZPZ = aerobus::zpz<907>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<783>, ZPZV<57>, ZPZV<905»;
           };   // NOLINT
04977      template<> struct ConwayPolynomial<911, 1> { using ZPZ = aerobus::zpz<911>; using type =
           POLYV<ZPZV<1>, ZPZV<894»; };   // NOLINT
04978      template<> struct ConwayPolynomial<911, 2> { using ZPZ = aerobus::zpz<911>; using type =
           POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; };   // NOLINT
04979      template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; };   // NOLINT
04980      template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17»; };   // NOLINT
04981      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
04982      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
04983      template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<894»; };   // NOLINT
04984      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
04985      template<> struct ConwayPolynomial<911, 9> { using ZPZ = aerobus::zpz<911>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<679>, ZPZV<116>, ZPZV<894»;
           };   // NOLINT
04986      template<> struct ConwayPolynomial<919, 1> { using ZPZ = aerobus::zpz<919>; using type =
           POLYV<ZPZV<1>, ZPZV<912»; };   // NOLINT
04987      template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
           POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; };   // NOLINT
04988      template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912»; };   // NOLINT
04989      template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7»; };   // NOLINT
04990      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
```

```
04991    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
04992    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
04993    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
04994    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
04995    template<> struct ConwayPolynomial<929, 1> { using ZPZ = aerobus::zpz<929>; using type =
     POLYV<ZPZV<1>, ZPZV<926>; };  // NOLINT
04996    template<> struct ConwayPolynomial<929, 2> { using ZPZ = aerobus::zpz<929>; using type =
     POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3>; };  // NOLINT
04997    template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<926>; };  // NOLINT
04998    template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3>; };  // NOLINT
04999    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
05000    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
05001    template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<926>; };  // NOLINT
05002    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
05003    template<> struct ConwayPolynomial<929, 9> { using ZPZ = aerobus::zpz<929>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<481>, ZPZV<199>, ZPZV<926>;
     };  // NOLINT
05004    template<> struct ConwayPolynomial<937, 1> { using ZPZ = aerobus::zpz<937>; using type =
     POLYV<ZPZV<1>, ZPZV<932>; };  // NOLINT
05005    template<> struct ConwayPolynomial<937, 2> { using ZPZ = aerobus::zpz<937>; using type =
     POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5>; };  // NOLINT
05006    template<> struct ConwayPolynomial<937, 3> { using ZPZ = aerobus::zpz<937>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<932>; };  // NOLINT
05007    template<> struct ConwayPolynomial<937, 4> { using ZPZ = aerobus::zpz<937>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<23>, ZPZV<585>, ZPZV<5>; };  // NOLINT
05008    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
05009    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
05010    template<> struct ConwayPolynomial<937, 7> { using ZPZ = aerobus::zpz<937>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<932>; };  // NOLINT
05011    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
05012    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
05013    template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
     POLYV<ZPZV<1>, ZPZV<939>; };  // NOLINT
05014    template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
     POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2>; };  // NOLINT
05015    template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<939>; };  // NOLINT
05016    template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2>; };  // NOLINT
05017    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
05018    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
05019    template<> struct ConwayPolynomial<941, 7> { using ZPZ = aerobus::zpz<941>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<939>; };  // NOLINT
05020    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
05021    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
05022    template<> struct ConwayPolynomial<947, 1> { using ZPZ = aerobus::zpz<947>; using type =
     POLYV<ZPZV<1>, ZPZV<945>; };  // NOLINT
05023    template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
     POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2>; };  // NOLINT
05024    template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<945>; };  // NOLINT
05025    template<> struct ConwayPolynomial<947, 4> { using ZPZ = aerobus::zpz<947>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2>; };  // NOLINT
05026    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
05027    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
05028    template<> struct ConwayPolynomial<947, 7> { using ZPZ = aerobus::zpz<947>; using type =
     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<945>; };  // NOLINT
05029    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
```

```
05030     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
05031     template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<950>; }; // NOLINT
05032     template<> struct ConwayPolynomial<953, 2> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3>; }; // NOLINT
05033     template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950>; }; // NOLINT
05034     template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<865>, ZPZV<3>; }; // NOLINT
05035     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
05036     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
05037     template<> struct ConwayPolynomial<953, 7> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<950>; }; // NOLINT
05038     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
05039     template<> struct ConwayPolynomial<953, 9> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<819>, ZPZV<316>, ZPZV<950>;
      }; // NOLINT
05040     template<> struct ConwayPolynomial<967, 1> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<962>; }; // NOLINT
05041     template<> struct ConwayPolynomial<967, 2> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5>; }; // NOLINT
05042     template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962>; }; // NOLINT
05043     template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5>; }; // NOLINT
05044     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
05045     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
05046     template<> struct ConwayPolynomial<967, 7> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<962>; }; // NOLINT
05047     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
05048     template<> struct ConwayPolynomial<967, 9> { using ZPZ = aerobus::zpz<967>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<512>, ZPZV<783>, ZPZV<962>;
      }; // NOLINT
05049     template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<965>; }; // NOLINT
05050     template<> struct ConwayPolynomial<971, 2> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<970>, ZPZV<6>; }; // NOLINT
05051     template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<965>; }; // NOLINT
05052     template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<527>, ZPZV<6>; }; // NOLINT
05053     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
05054     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
05055     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
05056     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
05057     template<> struct ConwayPolynomial<971, 9> { using ZPZ = aerobus::zpz<971>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<473>, ZPZV<965>;
      }; // NOLINT
05058     template<> struct ConwayPolynomial<977, 1> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<974>; }; // NOLINT
05059     template<> struct ConwayPolynomial<977, 2> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3>; }; // NOLINT
05060     template<> struct ConwayPolynomial<977, 3> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<974>; }; // NOLINT
05061     template<> struct ConwayPolynomial<977, 4> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<800>, ZPZV<3>; }; // NOLINT
05062     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
05063     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
05064     template<> struct ConwayPolynomial<977, 7> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<974>; }; // NOLINT
05065     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
05066     template<> struct ConwayPolynomial<977, 9> { using ZPZ = aerobus::zpz<977>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<740>, ZPZV<974>;
      }; // NOLINT
05067     template<> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<978>; }; // NOLINT
05068     template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5>; }; // NOLINT
```

```
05069     template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978»; };  // NOLINT
05070     template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5»; };  // NOLINT
05071     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
05072     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
05073     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
05074     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
05075     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
05076     template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
    POLYV<ZPZV<1>, ZPZV<985»; };  // NOLINT
05077     template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
    POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; };  // NOLINT
05078     template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985»; };  // NOLINT
05079     template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6»; };  // NOLINT
05080     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
05081     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
05082     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
05083     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
05084     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
05085     template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
    POLYV<ZPZV<1>, ZPZV<990»; };  // NOLINT
05086     template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
    POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; };  // NOLINT
05087     template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990»; };  // NOLINT
05088     template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7»; };  // NOLINT
05089     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
05090     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
05091     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
05092     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
05093     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
05094 #endif  // DO_NOT_DOCUMENT
05095 }  // namespace aerobus
05096 #endif  // AEROBUS_CONWAY_IMPORTS
05097
05098 #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 aerobus::ContinuedFraction

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

represents a continued fraction a0 + $\frac{1}{a_1 + \frac{1}{a_2 + \ldots}}$[https://en.wikipedia.org/wiki/Continued_↩ fraction](See in Wikipedia)

Template Parameters

...values	are int64_t

$<1, 1, 1>$ represents $1 + \frac{1}{\frac{1}{1}}$

10.32 PI_fraction::val

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

10.33 E_fraction::val

approximation of e -> 2.718...

approximation of e -> 2.718...

$<1, 1, 1>$ represents $1 + \frac{1}{\frac{1}{1}}$

Index